

PR #35233 完整报告

sgl-project/sglang

[AMD] Fix registered HiCache host pointer aliases

合并时间: 2026-09-15 06:49

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/35233>

执行摘要

- 一句话: 修复 HiCache registered 内存 CPU/GPU 指针域错配的 GPU 崩溃
- 推荐动作: 值得精读。核心看点是“registered host 内存双地址域”的认知与统一抽象: Python 侧 `make_kernel_ptr_table` 收敛了指针表构建, JIT 侧 `DeviceGuard` 用 `RAII` 保证多设备正确性, 测试覆盖 CUDA Graph 捕获 / 重放下的指针处理。对涉及 host 内存与自定义内核绑定的开发者有直接参考价值; 建议后续将“CPU VA 与 device alias 不得混用”的约束写入内存池模块文档。

功能与动机

PR body 明确指出: registered HiCache host allocation 在 CPU 与 GPU 侧可能具有不同的虚拟地址, MI355X 上 `hipDeviceAttributeCanUseHostPointerForRegisteredMem` 为 `false`, 把 CPU VA 传给自定义内核会触发 GPU memory fault, 而 runtime copy API 仍能接受该地址。受影响路径覆盖 AOT/JIT HiCache 传输、Mamba 传输、HiSparse 拷贝与 host-pool 指针表; 同时 CPU 侧 storage 与 disaggregation 元数据必须继续暴露原始 CPU VA。外部评论也印证了该问题: seokwoosong 在 NVIDIA/WSL2 (RTX 5090, CUDA 13.0) 独立复现同一 illegal memory access, yichiche 报告 InferenceX agent 模式下 `--hicache-io-backend kernel --hicache-mem-layout page_first` 出现 `hipErrorIllegalAddress` (Error code 700), 两者都指向同一指针域根因。

实现拆解

变更入口

问题根因是 registered host 内存存在 CPU 与 GPU 两个地址别名, 自定义内核必须使用 device alias。修复分五步落地:

1. AOT 层新增设备指针解析算子: 在 `python/sglang/kernels/aot/csrc/kvcacheio/transfer.cu` 实现 `get_device_accessible_ptr`, 按平台调用 `cudaHostGetDevicePointer` / `hipHostGetDevicePointer` 把 registered host 内存的 CPU 虚拟地址换算为当前设备可访问别名; 同时在 `python/sglang/kernels/aot/include/sgl_kernel_ops.h` 声明, 并在 `common_extension.cc` 与 `common_extension_rocm.cc` 注册为 `torch.ops.sgl_kernel.get_device_accessible_ptr`, Python 侧由 `python/sglang/kernels/aot/python/sgl_kernel/kvcacheio.py` 暴露薄封装。Runtime copy API 能容忍 CPU 地址, 但自定义内核没有地址翻译机制, 这是必须显式求别名的根本原因。

2. JIT 运行时引入设备守卫与别名解析：在 `python/sglang/kernels/jit/include/sgl_kernel/runtime.cuh` 新增 `DeviceGuard`（构造时记录原 device 并切换到目标设备、析构恢复，RAII 封装），并在 `hicache.cuh`、`hispase.cuh`、`transfer_mamba.cuh` 的 kernel launch 边界把直接使用 `tensor.data_ptr()` 的操作数替换为设备可访问别名。`DeviceGuard` 最初位于 `utils.cuh`，经 review 后按 DarkSharpness 建议移入 `runtime.cuh`。
3. 主机池统一指针表构建：`python/sglang/srt/mem_cache/pool_host/common.py` 新增 `make_kernel_ptr_table`，仅当 `host_memory_registered=True` 且目标设备为 cuda 时逐 tensor 求 device alias，否则原样取 `data_ptr()`。MHA（`mha.py` 的 `k_data_ptrs` / `v_data_ptrs`）、MLA（`mha.py` 的 `data_ptrs`）、K-only、DSV4 paged/state（`memory_pool_host.py`）、DSA（`dsa.py` 的 `index_k_data_ptrs`）全部切换为统一构建，保证“内核拿 device 地址、宿主拿 CPU 地址”两条线不混。
4. CPU 域元数据保持原始地址：`get_contiguous_buf_infos` 在 `mha.py` 与 `memory_pool_host.py` 中从读取 `data_ptrs` 表改为直接读取 `tensor.data_ptr()`，确保 disaggregation transfer engine 与 storage 层拿到的仍是 CPU VA；unregister、page metadata 路径不改动。
5. 回归测试配套：`test/registered/kernels/ops/kvcache/test_hicache_page_first_write_back.py` 新增 `_registered_host_pool` 构造器与两个用例——`test_registered_mmap_pointer_domains_and_all_layer_transfer` 验证指针域分离与 AOT 全层传输，`test_registered_mmap_page_first_kernel_operands_and_graph` 覆盖 MHA/MLA 在 `page_first` 布局下 AOT 传输与 CUDA Graph 捕获 / 重放。ROCm MI355X 上 4 组用例全部通过；CUDA 侧因无实机仅做 SM90 编译验证。

关键文件：

- `python/sglang/srt/mem_cache/pool_host/common.py`（模块 主机池；类别 source；类型 core-logic；符号 `make_kernel_ptr_table`）：新增 `make_kernel_ptr_table`，是 Python 侧 CPU/GPU 地址域分离的核心抽象，统一所有主机池指针表构建。
- `python/sglang/srt/mem_cache/pool_host/mha.py`（模块 主机池；类别 source；类型 core-logic）：MHA/K-only 主机池的 `k_data_ptrs` 与 `v_data_ptrs` 改用 `make_kernel_ptr_table`；Qwen3.5 GQA 层走此路径，是 InferenceX 崩溃修复的关键覆盖点。
- `python/sglang/srt/mem_cache/memory_pool_host.py`（模块 主机池；类别 source；类型 core-logic）：DSV4 paged/state 主机池的 `data_ptrs` 改用 `make_kernel_ptr_table`，且 `get_contiguous_buf_infos` 改回读 `data_ptr()` 保证 CPU 域元数据。
- `python/sglang/srt/mem_cache/pool_host/mha.py`（模块 主机池；类别 source；类型 core-logic）：MLA 主机池 `data_ptrs` 改用 `make_kernel_ptr_table`，`get_contiguous_buf_infos` 改回 CPU VA，DSV4 等 MLA 模型路径依赖此修复。
- `python/sglang/srt/mem_cache/pool_host/dsa.py`（模块 主机池；类别 source；类型 core-logic）：DSA index-K 指针表使用 device alias，覆盖 FP8 DSA 场景的 HiCache 路径。
- `python/sglang/kernels/aot/python/sgl_kernel/kvcacheio.py`（模块 内核绑定；类别 source；类型 core-logic；符号 `get_device_accessible_ptr`）：暴露 `get_device_accessible_ptr` 的 Python 包装，是 AOT 设备别名解析的入口。

- python/sglang/kernels/aot/csrc/kvcacheio/transfer.cu (模块 内核实现; 类别 other; 类型 core-logic) : AOT get_device_accessible_ptr 的 CUDA/ROCm 实现所在文件, 基于 cudaHostGetDevicePointer / hipHostGetDevicePointer。
- python/sglang/kernels/jit/include/sgl_kernel/runtime.cuh (模块 JIT 运行时; 类别 other; 类型 core-logic; 符号 DeviceGuard, device_accessible_ptr) : JIT 侧 DeviceGuard 与 device_accessible_ptr 所在文件, 是内核 launch 边界地址域解析的核心运行时。
- python/sglang/kernels/jit/csrc/kvcacheio/hicache.cuh (模块 JIT 内核; 类别 other; 类型 dependency-wiring) : HiCache JIT kernel launch 边界接入 device alias 解析, 写回 / 加载内核的直接覆盖点。
- python/sglang/kernels/jit/csrc/kvcacheio/hisparsed.cuh (模块 JIT 内核; 类别 other; 类型 dependency-wiring) : HiSparse 拷贝路径接入 DeviceGuard, review 中 DarkSharpness 在此质疑单设备假设。
- python/sglang/kernels/jit/csrc/kvcacheio/transfer_mamba.cuh (模块 JIT 内核; 类别 other; 类型 dependency-wiring) : Mamba 状态传输路径接入 device alias, Qwen3.5 线性注意力状态走此路径。
- test/registered/kernels/ops/kvcache/test_hicache_page_first_write_back.py (模块 回归测试; 类别 test; 类型 test-coverage; 符号 _registered_host_pool, test_registered_mmap_pointer_domains_and_all_layer_transfer, test_registered_mmap_page_first_kernel_operands_and_graph) : 新增 registered-mmap 回归测试, 覆盖指针域分离、AOT/JIT 传输与 CUDA Graph 捕获 / 重放, 是外部崩溃案例的验证载体。

关键符号: get_device_accessible_ptr, make_kernel_ptr_table, DeviceGuard, device_accessible_ptr, get_contiguous_buf_infos, test_registered_mmap_pointer_domains_and_all_layer_transfer, test_registered_mmap_page_first_kernel_operands_and_graph

关键源码片段

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

新增 make_kernel_ptr_table, 是 Python 侧 CPU/GPU 地址域分离的核心抽象, 统一所有主机池指针表构建。

```
def make_kernel_ptr_table(
    tensors: list[torch.Tensor],
    target_device: torch.device | str,
    *,
    host_memory_registered: bool,
) -> torch.Tensor:
    # 核心语义: registered host 内存同时在 CPU 与 GPU 地址域存在两个别名。
    # 内核必须拿到 GPU 可访问地址 (device alias), 而 CPU 侧元数据仍用
    # data_ptr() 原始地址; 这里按 host_memory_registered 与目标设备分流,
    # 保证 NPU / MUSA / 非 pinned 场景行为不变。
    device = torch.device(target_device)
    if host_memory_registered and device.type == "cuda":
```

```

# 延迟导入，避免在没有 sgl_kernel 的环境（如 NPU）里引入硬依赖
from sgl_kernel.kvcacheio import get_device_accessible_ptr

if device.index is None:
    device_index = torch.cuda.current_device()
else:
    device_index = device.index
pointers = [
    get_device_accessible_ptr(tensor, device_index) for tensor in tensors
]
else:
    pointers = [tensor.data_ptr() for tensor in tensors]
return torch.tensor(
    pointers,
    dtype=torch.uint64,
    device=device,
)

```

python/sglang/kernels/jit/include/sgl_kernel/runtime.cuh

JIT 侧 DeviceGuard 与 device_accessible_ptr 所在文件，是内核 launch 边界地址域解析的核心运行时。

```

// JIT 内核 launch 边界使用的设备守卫：registered host 指针的设备别名
// 依赖当前 CUDA/HIP 设备。先记录调用方原设备，再切换到目标设备，
// 用 RAII 保证 kernel 返回后恢复，避免多卡（含 DCP/PD 场景）串流。

```

```

class DeviceGuard {
public:
    explicit DeviceGuard(int target_device) {
#ifdef USE_ROCM
        RuntimeDeviceCheck(::hipGetDevice(&original_device_));
        if (original_device_ != target_device) {
            RuntimeDeviceCheck(::hipSetDevice(target_device));
            changed_ = true;
        }
#else
        RuntimeDeviceCheck(::cudaGetDevice(&original_device_));
        if (original_device_ != target_device) {
            RuntimeDeviceCheck(::cudaSetDevice(target_device));
            changed_ = true;
        }
#endif
    }

    ~DeviceGuard() {
        if (changed_) {
#ifdef USE_ROCM
            (void)::hipSetDevice(original_device_);
#else
            (void)::cudaSetDevice(original_device_);

```

```

#endif
    }
}

DeviceGuard(const DeviceGuard&) = delete;
DeviceGuard& operator=(const DeviceGuard&) = delete;

private:
    int original_device_ = 0;
    bool changed_ = false;
};

// 配套的 device_accessible_ptr 在 hicache.cuh / hisparse.cuh /
// transfer_mamba.cuh 的 launch 前把 registered host tensor 解析为
// 当前设备可访问别名；未注册内存与显存 tensor 原样返回 data_ptr()。

```

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

新增 registered-mmap 回归测试，覆盖指针域分离、AOT/JIT 传输与 CUDA Graph 捕获 / 重放，是外部崩溃案例的验证载体。

```

def test_registered_mmap_pointer_domains_and_all_layer_transfer() -> None:
    from sgl_kernel.kvcacheio import get_device_accessible_ptr

    # 用 registered mmap 主机内存构建 MLA host pool，验证地址域约束：
    # 1) 内核指针表 data_ptrs 必须等于 get_device_accessible_ptr 结果（device alias）；
    # 2) disaggregation 元数据 get_contiguous_buf_infos 必须返回原始 CPU VA；
    # 3) page 元数据仍指向 kv_buffer 的 CPU 地址。
    device_pool = MLATokenToKVPool(
        size=PAGE_SIZE * 4,
        page_size=PAGE_SIZE,
        kv_lora_rank=512,
        qk_rope_head_dim=64,
        dtype=torch.bfloat16,
        layer_num=NUM_LAYERS,
        device=DEVICE,
        enable_memory_saver=False,
    )
    host_pool = _registered_host_pool(
        MLATokenToKVPoolHost,
        device_pool=device_pool,
        layout="layer_first",
    )
    try:
        device_index = torch.cuda.current_device()
        kernel_ptrs = [
            get_device_accessible_ptr(tensor, device_index)
            for tensor in host_pool.data_refs
        ]
        assert host_pool.data_ptrs.cpu().tolist() == kernel_ptrs

```

```

raw_ptrs, _, _ = host_pool.get_contiguous_buf_infos()
assert raw_ptrs == [tensor.data_ptr() for tensor in host_pool.data_refs]
page_ptrs, _ = host_pool.get_page_buffer_meta(torch.arange(PAGE_SIZE))
assert page_ptrs[0] == host_pool.kv_buffer.data_ptr()

for layer_id in range(NUM_LAYERS):
    _fill_with_offset(device_pool.kv_buffer[layer_id], layer_id + 1)
    host_indices = _token_indices_for_pages(torch.tensor([0]))
    device_indices = _token_indices_for_pages(torch.tensor([1]))
    host_pool.can_use_jit = False
    host_pool.backup_from_device_all_layer(
        device_pool, host_indices, device_indices, "kernel"
    )
    torch.cuda.synchronize()
    for layer_id in range(NUM_LAYERS):
        _assert_pages_equal(
            host_pool.data_refs[layer_id],
            device_pool.kv_buffer[layer_id],
            torch.tensor([0]),
            torch.tensor([1]),
        )
    )
finally:
    host_pool.destroy()

```

评论区精华

核心审查者 DarkSharpness 与作者 AMD-yanfeiwang 的交锋集中在 JIT 侧设备切换的必要性上：

DarkSharpness: Why do we need to change device here? Typically we're always on the same device. (针对 `DeviceGuard`) DarkSharpness: Why do we need this. Typically we only have one device for each process. (针对 `hisparse.cuh` 中的 `DeviceGuard device_guard(kernel_device.device_id)`)

作者保留该设计并逐一回应“Resolved. PTAL”，说明在多卡、DCP/PD 场景下“单进程单设备”假设不成立，kernel launch 边界必须保证当前设备正确。另有两条风格类建议：将 `DeviceGuard` 从 `utils.cuh` 移到 `runtime.cuh`、重命名 `device_accessible_ptr`，均已解决。最终 DarkSharpness 给出“LGMT on kernel side.”的 APPROVED。

Issue 评论区的价值更高：seokwoosong 在 NVIDIA/WSL2 独立复现同一问题并用 `cudaHostGetDevicePointer` 别名修复；jiejingzhangamd 完成了 Qwen3.5 端到端验证与三臂基准对比，确认修复无精度回退，并指出速度提升（291 → 338 tok/s）主要来自切换 `kernel` 后端而非本 PR 本身。

- InferenceX agent 模式 Error code 700 是否被本 PR 覆盖 (question): jiejingzhangamd 在 MI355X 上用 main + #35233 完成 Qwen3.5 端到端验证，无 code 700、无非法访存，HiCache 计数器确认 kernel 路径实际运行。

- JIT launch 边界切换设备的必要性 (design): 多卡 /DCP 场景下当前设备不保证与目标 tensor 一致, launch 前切换并恢复是必要的; 设计保留。
- hisparse.cuh 中 DeviceGuard 的单设备假设 (design): 保留 DeviceGuard 封装, 作者标记已解决。
- DeviceGuard 放置位置 (style): 已移到 runtime.cuh, 与 runtime 设备检查逻辑同层组织。
- device_accessible_ptr 命名 (style): 命名已调整, 避免与 AOT 侧 get_device_accessible_ptr 混淆。

风险与影响

- 风险:
 1. CUDA 实机覆盖不足: 本 PR 在 NVIDIA 上仅做 SM90 编译验证, 无运行时 CI 回归; 虽有外部 WSL2 独立复现佐证, 但全量 CUDA 行为风险仍存。
 2. 指针域语义易误用: data_ptrs 的语义从 CPU VA 变为 device alias, 未来任何新消费方若再把它当 CPU 地址 (例如新增 get_contiguous_buf_infos 之外的调用点) 会产生难排查的非法访存; 当前通过直接读 data_ptr() 规避 disaggregation 路径, 但属于隐式约定, 需要文档固化。
 3. 多设备假设: make_kernel_ptr_table 在 device.index is None 时依赖 torch.cuda.current_device(), JIT 侧 DeviceGuard 只切换不校验目标设备合法性, 错误设备会得到非法 alias。
 4. 性能未实测: PR body 声明不改传输算法, 但 AOT/JIT 路径在 launch 前多了一次指针解析与设备切换; 指针表在 pool 初始化时构建, 热路径影响预计很小, 但缺少 benchmark 数据。
 5. 平台差异: NPU/MUSA 走 alloc_with_pin_memory 且非 cuda 设备, 不会进入 device alias 分支, 行为保持不变; 若未来 NPU 引入 registered 内存路径需单独适配。- 影响: 影响面覆盖所有启用 Host KV Cache (HiCache) 的 CUDA/ROCm 部署, 尤其是 MI355X 与 WSL2 环境: kernel 传输后端与 page_first 布局的 load/write-back 不再触发 GPU memory fault 或 illegal memory access。Qwen3.5 (GQA 走 MHA 主机池、线性注意力走 MambaPoolHost)、DeepSeek V4 (paged/state 池) 与依赖 HiSparse 的模型路径均被覆盖。对 CPU 侧 storage 与 disaggregation 元数据无影响。团队侧, 该修复解除了 AMD 平台 HiCache kernel 后端的稳定阻塞, 为后续 #37152 等性能优化铺路, 并沉淀了 registered-mmap + CUDA Graph 的回归测试模式。- 风险标记: 核心路径变更, 跨模块多后端影响, CUDA 仅编译验证, 地址域语义易误用, 性能未实测

关联脉络

- PR #37413 [AMD][DSV4] feat: enable fp8 two-pool unified_kv on gfx950: 同属 AMD + memory-pool + DSV4 host KV 演进线, 本 PR 的 DSV4 paged/state 池指针修复是其前置正确性保障。
- PR #37382 [NPU] Support DSV4 host memory cache management: 同文件域 (memory_pool_host.py、unified_radix_cache.py) 的 host 内存池演进, 本 PR 的 CPU/GPU 地址域问题在该域同样存在。

- PR #39162 [HiCache][LoRA] Simplify decode offload hash inputs: HiCache + disaggregation offload 路径, 依赖本 PR 保证的 CPU 地址元数据语义。
- PR #39358 [AMD] Align Qwen3.5 MI355X cookbook with AttnFP8-V2 and HiCache direct / page_first_direct: Qwen3.5 MI355X HiCache 后端配置演进, 与本 PR 讨论中覆盖的 Qwen3.5 路径直接相关。
- PR #37152 (评论中提及的关联 PR, 标题未在材料中提供): jiejingzhangamd 的三臂基准对比中作为第三臂出现 (baseline + #35233 + #37152), 与 kernel 后端优化相关。