PR #34167 完整报告

sgl-project/sglang

[DSA] Fix top-k v2 dropping non-primary ranks' output on CUDA 13.1+ (root cause for #33835)

合并时间：2026-08-10 10:31

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34167

执行摘要

- 一句话：修复 DSV4 top-k 在 CUDA 13.1+ 丢非 primary rank 输出
- 推荐动作：值得精读。这是少见的“工具链版本敏感型编译器错误”调试案例：作者系统排除 9 类假设、用 slot-diff 与插桩把根因定位到单一指针变量携带两个地址空间，方法论本身有很高的迁移价值；__builtin_assume 作为 load-bearing 约束而非优化提示的用法也值得记录。建议阅读时重点关注 topk_impl.cuh 中两条散射路径的拆分注释、topk_v2.cuh 中 __builtin_assume 的依赖说明，以及 v1 模块 STB_GNU_UNIQUE 符号合并的分析。后续可跟进 sm_100a 运行时验证与 CI nvcc 版本保护。

功能与动机

PR 正文指出：DeepSeek-V4 DSA decode 会从 fused small-batch cluster 内核拿到损坏的 top-k，下游 sparse attention 将其解引用为垃圾 KV 索引，即 #33835 报告的 illegal memory access。关键洞察是触发条件为 CUDA 工具链版本而非 GPU 架构：同一份源码、同一块 H200（sm_90a），用 nvcc 12.9/13.0 构建正确，而 13.1/13.2/13.3 构建错误。v1 的动机来自第二个 commit：-DSGL_TOPK 宏把 topk 喂给 constexpr 而非模板参数，导致多个 JIT 模块导出同名 mangled 符号并被 loader 合并，后加载模块跳过 cudaFuncSetAttribute opt-in，最终在 64 KB 动态共享内存启动时报 invalid argument。

实现拆解

变更入口是 TopKCluster::forward（python/sglang/kernels/jit/include/sgl_kernel/deepseek_v4/topk_impl.cuh），随后沿 v2 内核与 v1 模块两条线补齐：

1. 根因定位与散射路径拆分（topk_impl.cuh）：原实现先通过 cur_out = is_primary ? problem.out : smem->tmp_out 选定 phase-3 散射目标，两个地址空间流入同一指针变量后触发 cicc 13.1+ 错编译。作者用 slot-diff 证实丢失的 383 个槽位恰好对应七个 peer，并逐一排除 9 类候选假设。修复把 primary 与非 primary 拆成两条独立散射循环：primary 直接 problem.emit(pos, idx)，非 primary 先写 smem->tmp_out、在 cluster.sync() 后再 problem.emit，并在源码注释中明确禁止合并回一个指针变量。
2. v2 fused-cluster 路径加固（topk_v2.cuh）：移除 #32910 引入的 peer_problem 拷贝，改为 elected worker 直接 problem.out = cluster.map_shared_rank(topk_indices, worker_rank)，在 problem_transform 前加 __builtin_assume(problem.out == topk_indices)，把 block-local 指针约束显式化；该假设是 load-bearing，去掉即复现 #32830 的 cicc 段错误。非 fused 路径统一为 early return + PDLWaitPrimary +

__syncthreads 的顺序。

3. v1 运行时 topk（topk_v1.cuh 与 topk.py）：删除 -DSGL_TOPK 宏，引入 kMaxTopK = 1024 与 TopKParams::topk 运行字段；kTopKBlockSize 固定为 kMaxTopK，保证 histogram 初始化与 run_cumsum() 覆盖到 RADIX + 1 个线程；naive_transform、radix_topk、topk_transform_kernel 全部改为运行时 topk。Python 侧 _jit_topk_v1_module() 去掉 topk 参数与 per-k 命名，成为单一无参模块，规避 STB_GNU_UNIQUE 符号合并。
4. 验证与配套：本 PR 未新增测试文件，验证以扩展套件方式进行：157 行形状（#33835 报告形状 + 边界扫描 + register/streaming/persistent 对照）覆盖 5 个稳定 nvcc 小版本与 k ∈ {512, 1024, 2048}，另有 1500 次随机压力、500 次 CUDA-graph 重放、v1 的 80 组 k 用例（含 1/31/255/512/1023/1024 等边界）。#33835 已有的边界回归测试保留。作者指出CI构建cu130是干净单元，无法覆盖此类工具链问题，建议引入工具链版本守卫。

关键文件：

- python/sglang/kernels/jit/include/sgl_kernel/deepseek_v4/topk_impl.cuh（模块 内核；类别 source；类型 core-logic；符号 TopKCluster::forward）：TopKCluster::forward 是核心修复点：将 primary/ 非 primary 两条 phase-3 散射路径拆分，杜绝同一指针变量携带 DSMEM 与 CTA 两种地址空间，直接消除 CUDA 13.1+ 下静默丢弃非 primary rank 输出的根因。
- python/sglang/kernels/jit/csrc/deepseek_v4/topk_v2.cuh（模块 内核；类别 source；类型 core-logic；符号 topk_small_batch_kernel）：fused-cluster 路径的二次加固：删除 #32910 的 peer_problem 拷贝，用 load-bearing 的 __builtin_assume 把 block-local 指针约束显式化，避免 cicc 13.1+ 段错误。
- python/sglang/kernels/jit/csrc/deepseek_v4/topk_v1.cuh（模块 内核；类别 source；类型 core-logic；符号 topk_transform_kernel, radix_topk, naive_transform, TopKParams）：v1 从编译期 topk 宏改为运行时 topk：引入 kMaxTopK 与 TopKParams::topk，修复多模块同名符号被 loader 合并导致 64 KB DSMEM 启动失败，同时保证 histogram 线程覆盖。
- python/sglang/kernels/ops/attention/dsv4/topk.py（模块 JIT 加载；类别 infra；类型 infrastructure；符号 _jit_topk_v1_module, topk_transform_512）：JIT 模块加载配套改动：_jit_topk_v1_module 去掉 topk 参数与 -DSGL_TOPK 宏，收敛为单一模块，解决 STB_GNU_UNIQUE 符号合并导致的第二个模块启动失败。

关键符号：TopKCluster::forward, topk_small_batch_kernel, _jit_topk_v1_module, topk_transform_kernel, radix_topk, naive_transform, topk_transform_512

关键源码片段

python/sglang/kernels/jit/include/sgl_kernel/deepseek_v4/topk_impl.cuh

TopKCluster::forward 是核心修复点：将 primary/ 非 primary 两条 phase-3 散射路径拆分，杜绝同一指针变量携带 DSMEM 与 CTA 两种地址空间，直接消除 CUDA 13.1+ 下静默丢弃非 primary rank 输出的根因。

```
// TopKCluster::forward 的 phase-3 候选收集：修复后 primary 与非 primary
```

```
// 分支各自独立散射，不再共用一个 cur_out 指针变量，否则 DSMEM 与 CTA
// 两个地址空间会同时流入单个指针，触发 cicc 13.1+ 的错误编译。
if (!is_primary) {
    // 非 primary rank：候选先暂存到 block-local 的 smem->tmp_out。
    // 注意：不要合并回 "cur_out = is_primary ? problem.out : smem->tmp_out"。
    // problem.out 可能是 shared::cluster（DSMEM）别名，而 tmp_out 是 shared::cta；
    // cicc 13.1+ 会把 block-local 分支错误地下沉，导致 tmp_out 全 0，
    // 后续 phase 3.5 再把 0 原样复制到正确的 DSMEM 地址（#33835 的根因）。
    for_each_input(problem.in, local_seq_len, [&](float val, uint32_t local_idx) {
        const auto idx = chunk_start + local_idx;
        if (val >= v_hi) {
            const auto pos = atomicAdd(&smem->count_gt, 1);
            if (pos < topk) [[likely]] {
                smem->tmp_out[pos] = idx;
            }
        } else if (val >= v_lo) {
            const auto count_eq = atomicAdd(&smem->count_eq, 1);
            if (count_eq < kMaxNumTie) [[likely]] {
                smem->tie.values[count_eq] = {val, idx};
            }
        }
    });
    __syncthreads();
    const auto local_above_count = smem->count_gt;
    const auto local_equal_count = min(smem->count_eq, kMaxNumTie);
    // 省略跨 rank 前缀和：这里得到 start_gt_local / start_eq_local
    cluster.sync();
    const auto start_write = start_gt_local;
    const auto num_write = local_above_count;
#pragma unroll
    for (uint32_t i = 0; i < kTopKItems; ++i) {
        if (const auto t = tx + i * kBlockSize;
            t < num_write && start_write + t < topk) {
            problem.emit(start_write + t, smem->tmp_out[t]);
        }
    }
} else {
    // primary rank：直接散射进 problem.out（最终输出位置，可能是 DSMEM）
    for_each_input(problem.in, local_seq_len, [&](float val, uint32_t local_idx) {
        const auto idx = chunk_start + local_idx;
        if (val >= v_hi) {
            const auto pos = atomicAdd(&smem->count_gt, 1);
            if (pos < topk) [[likely]] {
                problem.emit(pos, idx);
            }
        } else if (val >= v_lo) {
            const auto count_eq = atomicAdd(&smem->count_eq, 1);
            if (count_eq < kMaxNumTie) [[likely]] {
                smem->tie.values[count_eq] = {val, idx};
```

```
            }
        }
    });
    cluster.sync();
    // Phase 4: 处理并列值 handle_tie（省略尾部）
}
```

python/sglang/kernels/jit/csrc/deepseek_v4/topk_v2.cuh

fused-cluster 路径的二次加固：删除 #32910 的 peer_problem 拷贝，用 load-bearing 的 __builtin_assume 把 block-local 指针约束显式化，避免 cicc 13.1+ 段错误。

```
// topk_small_batch_kernel 的 fused-cluster 路径：修复后的问题转发与回读
if (problem.seq_len <= kReg4MaxSeqLen) {
    if (blockIdx.y != worker_rank) return;
    Register4::forward<kPDL>(problem, &smem);
    device::PDLWaitPrimary<kPDL>();
    __syncthreads();
} else if (problem.seq_len <= params.cluster_floor) {
    if (blockIdx.y != worker_rank) return;
    Streaming::forward<kPDL>(problem, &smem);
    device::PDLWaitPrimary<kPDL>();
    __syncthreads();
} else {
    auto cluster = cooperative_groups::this_cluster();
    // 直接让 elected rank 的 problem.out 指向自己的 DSMEM 映射，
    // 取代 #32910 引入的 peer_problem 拷贝方案。
    problem.out = cluster.map_shared_rank(topk_indices, worker_rank);
    Cluster::forward<kPDL>(problem, &smem); // 写 peer 的 output shared memory
    device::PDLWaitPrimary<kPDL>();
    cluster.sync();
    if (blockIdx.y != worker_rank) return;
}
// 只有 elected worker 能走到这里，且 problem.out 已被映射到本 block 的 buffer。
// 这个 __builtin_assume 是 load-bearing 而非优化：去掉它会在 CUDA 13.1+ 的
// sm_90a 上复现 cicc 段错误（issue #32830）。
__builtin_assume(problem.out == topk_indices);
problem_transform(problem, params.get_output_ptr(blockIdx.x));
```

python/sglang/kernels/ops/attention/dsv4/topk.py

JIT 模块加载配套改动：_jit_topk_v1_module 去掉 topk 参数与 -DSGL_TOPK 宏，收敛为单一模块，解决 STB_GNU_UNIQUE 符号合并导致的第二个模块启动失败。

```
# v1 的 topk 从编译期常量改为运行期参数：一个 JIT 模块服务所有 k。
# 之前用 -DSGL_TOPK 宏按 k 各编一个模块，由于宏喂给 constexpr 而非模板
# 参数，多个模块导出完全同名的 mangled 符号；setup_kernel_smem_once 的
# 函数局部 static 以 STB_GNU_UNIQUE 发射，被 loader 跨对象合并，
# 后加载的模块跳过 cudaFuncSetAttribute opt-in，最终在 64 KB 动态共享
# 内存启动时报 "invalid argument"（bench_topk.py 同进程 sweep k 会踩中）。
@cache_once
```

```
def _jit_topk_v1_module():
    args = make_cpp_args(is_arch_support_pdl())
    return load_jit(
        make_name("topk_v1"),
        *args,
        cuda_files=["deepseek_v4/topk_v1.cuh"],
        cuda_wrappers=[("topk_transform", f"TopKKernel<{args}>::transform")],
    )

# 调用方（如 topk_transform_512）不再按 k 选择模块，统一：
# module = _jit_topk_v1_module()
# module.topk_transform(...)
```

评论区精华

维护者 BBuf 的审批意见只有一句："It's a UB fix, LGTM."，一次性通过，未要求新增改动。更有价值的讨论沉淀在 PR 正文：作者记录了 9 类被排除的假设（enable_smem_spilling、__restrict__、强制 st.u32 地址空间、volatile 读取、barrier.cluster.arrive.release/wait.acquire、__threadfence() 等），并用 slot-diff 与 read-back 插桩把根因钉死在"单一指针变量携带两个地址空间"上；同时还纠正了 #33835 的归因——并非 Hopper 专属、也并非仅 32K floor 上方少量行，而是所有 fused small-batch cluster 形状（batch <= 30 且 seq_len > cluster_floor）都受影响，floor 只是决定路径可达性的阈值。作者在 Caveats 中主动披露：sm_100a 只有编译验证、CI cu130 无法捕获本类问题、v1 剩余 exported-weak-symbol 风险未处理。

- UB 修复评审与根因论证 (correctness): 批准合并。作者在 Caveats 中承认 sm_100a 仅编译验证、CI 的 cu130 构建无法覆盖该工具链问题类别。

风险与影响

- 风险：
  1. 工具链版本敏感：修复依赖编译器正确区分两条地址空间路径，cicc 后续版本仍可能回归；CI 构建 cu130 是干净单元，目前无法自动捕获，建议增加 13.1+ 构建或工具链守卫。
  2. Blackwell 未实测：sm_100a 仅编译验证，fused-cluster 路径在 Blackwell 上的运行时行为仍待确认。
  3. __builtin_assume 的隐式契约（topk_v2.cuh）：该假设一旦被后续重构删除，会以 cicc 段错误而非明显报错的方式复发，需要靠代码注释与 #32830 关联维护。
  4. v1 符号可见性隐患未除：导出符号的 weak 属性仍可能让多模块进程静默失败，作者建议以 hidden visibility 编译 JIT 模块作为通用修复。
  5. 影响面扩展：修复覆盖所有 fused small-batch cluster 形状而非仅 #33835 报告区间，属于核心 decode 路径；persistent-pool 路径不受影响。性能无回退，fused-cluster 路径获得 1.3%~2.9% 提升。 - 影响：用户与部署侧：所有使用 DeepSeek-V4 DSA 且以 nvcc 13.1+ 构建的服务会从随机非法内存访问或错误输出中恢复；CUDA 12.9/13.0 用户无行为变化。系统侧：消除了一处静默数据损坏源，top-k 输出恢复正确后，下游

sparse attention 不再收到垃圾 KV 索引。工程侧：确立了 JIT 内核按 nvcc 小版本建立回归矩阵的需求，并形成“不同地址空间的指针不得合并进同一变量”的代码审查约定。影响范围集中在 DSV4 fused small-batch cluster 路径与 v1 JIT 模块加载路径，不涉及 persistent-pool 路径。 - 风险标记：工具链版本敏感，核心推理路径，仅编译验证 Blackwell, JIT 符号可见性隐患，CI 无法覆盖 13.1+

关联脉络

- PR #34189 [DSV4] Fix silent KV corruption when speculative draft tokens > 4: 同属 DeepSeek-V4 内核级正确性修复线：本 PR 修复 top-k 输出损坏，34189 修复压缩环 KV 写坏，两者都防止静默数据损坏向下游传播。
- PR #26671 [JIT Kernel][DSv4] Optimize epilogue of c128: 同一 DeepSeek-V4 JIT kernel 模块的近期演进，说明该模块持续在正确性（本 PR）与性能（26671）上迭代。