PR #46203 完整报告

vllm-project/vllm

[Bugfix][ROCm] Fix cumem sleep and teardown

合并时间：2026-06-24 02:45

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/46203

执行摘要

- 一句话：修复 ROCm cumem sleep 内存释放和销毁崩溃
- 推荐动作：该 PR 修复了关键的内存泄漏和崩溃问题，值得所有 ROCm 用户升级。建议精读 release_pools 的两阶段释放模式，以及 C++ 层中处理睡眠分配的重入逻辑。此外，讨论中涉及的 HandleType 类型适配和 is_asleep 安全措施值得代码审查时借鉴。

功能与动机

本 PR 修复了 tests/basic_correctness/test_mem.py 在 MI300X 上暴露的两个 ROCm cumem sleep 模式问题：

1) sleep 模式在 hipMemUnmap/hipMemRelease 后由于虚拟地址范围保留，物理 VRAM 未实际释放； 2) 正常解释器关闭时，销毁 kept-alive 的 cumem MemPool 对象可能导致崩溃，因为其插拔分配器包装已不再存活。如 PR body 所述：'Fix two ROCm cumem sleep-mode issues found in tests/basic_correctness/test_mem.py on MI300X'。

实现拆解

1. C++ 扩展修改物理释放流程：在 csrc/cumem_allocator.cpp 中，修改 reserve_rocm_address 以接受可选的地址参数，用于在释放后重新保留相同占位符。在 python_unmap_and_release 中，释放虚拟地址后立即调用 reserve_rocm_address 保留原地址作为占位符，从而真正回收物理内存（ROCm 行为要求）。同时在 my_free 中增加对空 chunk 列表的处理（睡眠分配），跳过物理释放操作，只释放虚拟地址。
2. Python 分配器安全关闭：在 vllm/device_allocator/cumem.py 中新增 release_pools 方法，分两阶段释放 MemPool 和 CUDAPluggableAllocator：先清除 mem_pools 引用并触发 GC，再清除 allocators，避免 MemPool 析构时调用已释放的插拔分配器。通过 _shutdown_singleton 静态方法并在 get_instance 中通过 atexit 注册，确保解释器退出时调用。同时添加 close 别名。在 sleep 方法中为每个分配设置 data.is_asleep = True，以便在 my_free 中正确识别睡眠中的分配。
3. 数据结构与类型适配：在 vllm/device_allocator/__init__.py 中，将 HandleType 定义为 tuple[int, int, int, list[int] | int] 以支持 ROCm 上句柄为列表的情况。在 AllocationData 中添加 is_asleep 标志。
4. 集成 shutdown 与库查找优化：在 vllm/v1/worker/gpu_worker.py 的 shutdown 方法中调用 CuMemAllocator.instance.release_pools()，确保正常关闭路径释放 pools。在

vllm/distributed/device_communicators/cuda_wrapper.py 中简化 __init__，根据平台直接查找对应库（libamdhip64 for ROCm, libcudart otherwise），避免先尝试不存在的库。

5. CI 测试启用：在 .buildkite/test_areas/basic_correctness.yaml 中添加 AMD 测试镜像配置，在 .buildkite/test-amd.yaml 中调整标签，使基本正确性测试在 AMD 上运行。

关键文件：

- vllm/device_allocator/cumem.py（模块 设备分配器；类别 source；类型 dependency-wiring；符号 _shutdown_singleton, release_pools, close）：核心修复，添加 atexit 注册的 shutdown 方法和两阶段 release_pools，解决解释器关闭时触发纯虚函数调用崩溃，并跟踪睡眠分配状态。
- csrc/cumem_allocator.cpp（模块 C 扩展；类别 source；类型 core-logic）：核心 C++ 扩展修改，实现 ROCm 上真正回收物理 VRAM 的地址释放与重新保留占位符逻辑。
- vllm/distributed/device_communicators/cuda_wrapper.py（模块 分布式通信；类别 source；类型 core-logic）：优化库查找路径，避免在 ROCm 上先搜索不存在的 libcudart，提高初始化鲁棒性。
- vllm/v1/worker/gpu_worker.py（模块 工作节点；类别 source；类型 dependency-wiring）：在 shutdown 方法中显式调用 release_pools，确保正常关闭路径释放 cumem pools。
- vllm/device_allocator/__init__.py（模块 分配器接口；类别 source；类型 dependency-wiring）：定义 HandleType 为联合类型以支持 ROCm，添加 AllocationData.is_asleep 字段。
- .buildkite/test_areas/basic_correctness.yaml（模块 CI 配置；类别 config；类型 configuration）：启用 AMD 测试镜像，使基本正确性测试在 AMD 上运行。
- .buildkite/test-amd.yaml（模块 CI 配置；类别 config；类型 configuration）：调整 CI 标签以配合新测试镜像。

关键符号：_shutdown_singleton, release_pools, close, reserve_rocm_address, python_unmap_and_release, sleep

关键源码片段

vllm/device_allocator/cumem.py

核心修复，添加 atexit 注册的 shutdown 方法和两阶段 release_pools，解决解释器关闭时触发纯虚函数调用崩溃，并跟踪睡眠分配状态。

```
def release_pools(self) -> None:
    """Drop Python references to MemPool/pluggable allocators eagerly.

    A cumem ``MemPool`` outlives the ``use_memory_pool`` context (a strong
    reference is kept in ``allocator_and_pools`` to work around
    pytorch/pytorch#146431), and a captured CUDA graph can keep it alive
    longer still. ``MemPool`` only holds a non-owning pointer to the
    allocator, whose owning reference lives in the Python
    ``CUDAPluggableAllocator``. If both are instead dropped during
    interpreter shutdown, GC may finalize the allocator first; the eventual
    ``~MemPool`` -> ``emptyCache`` -> ``release_block`` then makes a virtual
```

```
    call into the freed allocator -- aborting the process with "pure virtual
    method called" (pytorch/pytorch#145168).

    Release the kept-alive pools before interpreter finalization, and keep
    the pluggable allocator wrappers alive while MemPool destructors run.
    This is safe to call more than once.
    """
    if not self.allocator_and_pools:
        return

    pool_entries = list(self.allocator_and_pools.values())
    self.allocator_and_pools.clear()

    mem_pools = [entry[0] for entry in pool_entries]
    allocators = [entry[1] for entry in pool_entries]
    pool_entries.clear() # 释放 pool_entries 本身

    # Phase 1: 在 allocators 保持强引用时先清除 MemPool 引用
    mem_pools.clear()
    gc.collect() # 触发 MemPool 析构，此时 CUDAPluggableAllocator 仍存活

    # Phase 2: 安全释放 allocator wrappers
    allocators.clear()
```

csrc/cumem_allocator.cpp

核心 C++ 扩展修改，实现 ROCm 上真正回收物理 VRAM 的地址释放与重新保留占位符逻辑。

```
  // ROCm 专用：释放物理内存后立即重新保留虚拟地址占位符
  if (error_code == no_error) {
      // 释放虚拟地址：物理内存在 ROCm 上只有在虚拟地址释放后才真正回收
      CUDA_CHECK(cuMemAddressFree(d_mem_ptr, recv_size));
      if (error_code == no_error) {
          CUdeviceptr reserved = 0;
          // 立即重新保留相同地址作为空占位符，防止其他分配器使用
          CUDA_CHECK(reserve_rocm_address(&reserved, recv_size,
                                          /*alignment=*/0, d_mem_ptr));
          // 确保返回地址与原地址一致
          if (reserved != d_mem_ptr) {
              cuMemAddressFree(reserved, recv_size);
              error_code = CUDA_ERROR_INVALID_VALUE;
          }
      }
  }
  // 之后函数返回错误状态给 Python 层
```

评论区精华

- HandleType 类型适配：mawong-amd 指出 HandleType 在 ROCm 上应为 list[int]，而非当前统一类型。作者随后修正为联合类型 list[int] | int，但在 mypy 下仍存在问题，最终采用

无条件联合类型以避免条件类型别名限制。

- is_asleep 安全性分析：depthfirst-app[bot] 指出如果 C 扩展的 unmap_and_release 成功但后续重新保留步骤失败，is_asleep 标记不会被设置，可能导致后续 free 重复释放物理句柄。作者在提交 34e229c 中修复，将 unmap_and_release 包裹在 try/finally 中确保 is_asleep 始终设置。
- YAML 缩进错误：mawong-amd 指出 CI 配置的镜像块缩进错误，作者在后续提交中更正。
  - HandleType 在 ROCm 上应为 list[int] (correctness): 作者修正为联合类型 list[int] | int ，后改为无条件联合类型以避免 mypy 限制。
  - C 扩展异常处理导致 is_asleep 未设置 (correctness): 作者在提交 34e229c 中通过 try/finally 确保 is_asleep 始终设置。
  - CI YAML 缩进错误 (style): 作者在后续提交中修正了缩进。

风险与影响

- 风险：
  - 重新保留占位符失败：在 python_unmap_and_release 中，重新保留虚拟地址可能失败（如内存不足），此时物理块已释放但虚拟地址被释放，可能导致后续错误。尽管已用 try/finally 设置 is_asleep，但若保留失败，占位符缺失可能影响 wake_up 时的重新映射。
  - 两阶段释放依赖 GC：release_pools 依赖 GC 回收时机，如果 mem_pools 在 allocators 之后被间接引用，可能导致析构乱序。目前设计通过分步 clear 并调用 gc.collect() 降低风险，但在循环引用或复杂图景下可能不够鲁棒。
  - atexit 注册副作用：在 atexit 中注册单例方法，在子解释器或 fork 场景下可能导致意外行为或重复注册。
  - C 扩展分支处理：my_free 中根据 num_chunks 分支处理睡眠分配，若逻辑错误可能导致内存泄漏或双释放。
- 影响：
  - 用户：ROCm 用户使用 sleep 模式（如多 GPU 空闲时释放内存）将获得正确的内存回收，并且不再遇到程序退出时的崩溃。NV GPU 用户无行为变化。
  - 系统：修改涉及设备分配器、C 扩展、worker shutdown，影响面中等。在正常关闭和异常关闭路径均增加了释放逻辑。
  - 团队：该 PR 修复了长期存在的 MI300X 测试问题，提升了 CI 稳定性，方便后续开发。
  - 风险标记：ROCm sleep 内存释放，解释器关闭顺序依赖，C 扩展异常处理

关联脉络

- PR #43022 相关 issue/PR 引用了验证内存泄漏的复现步骤：本 PR 使用 #43022 中的复现命令验证内存泄漏修复结果。
- PR #46401 [Bugfix] Fix unrelated test regression on AMD: 该 PR 修复了本 PR 在 CI 中暴露的另一个无关测试回归。