PR #48526 完整报告

vllm-project/vllm

[ROCm] Re-enable cudagraph memory profiling, captured on the current stream

合并时间：2026-07-15 23:03

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/48526

执行摘要

- 一句话：重新启用 ROCm cudagraph 内存分析并修复吞吐回归
- 推荐动作：建议精读该 PR，它展示了 GPU 内存流分配对性能影响的深刻分析，以及一个优雅的修复——通过调整捕获流（而非重写内存管理）来解决问题。审查在线讨论（特别是关于 AITER 缓冲区所有权和 torch 缓存的深入技术分析）非常有价值。

功能与动机

之前的临时修复 #48440 禁用了 ROCm 上的 cudagraph 内存分析，导致 decode 吞吐量下降约 20%。该 PR 旨在重新启用该功能并消除吞吐量回归，同时提供正确的内存估计。PR 描述中提到了关联 Issue #48453。

实现拆解

1. 扩展 graph_capture 上下文管理器：在 vllm/distributed/parallel_state.py 中，graph_capture 函数新增了可选的 graph_capture_context 参数。如果提供了该参数，则使用该上下文中的流；否则回退到默认行为（创建新的侧流）。这样可以灵活地控制捕获流。
2. 在 ROCm 上使用当前流进行临时分析：在 vllm/v1/worker/gpu_model_runner.py 的 profile_cudagraph_memory() 方法中，新增了 ROCm 专属逻辑：创建一个使用当前流的 GraphCaptureContext 实例，并在调用 graph_capture 时传入。这使得临时分析图形在运行时流上进行捕获，避免在侧流分配池中滞留 AITER 临时缓冲区（约 76 MiB）。CUDA 上的行为保持不变（cap_ctx=None），因为 CUDA 的当前流是传统默认流，无法在此开始捕获。
3. 重新启用 ROCm 的分析入口：在 vllm/v1/worker/gpu_worker.py 的 determine_available_memory() 方法中，将条件从 current_platform.is_cuda() 改为 current_platform.is_cuda_alike()，使 ROCm 能够执行 profile_cudagraph_memory()。同时移除了之前对 ROCm 不支持的误导性注释，并更新了相关说明。

关键文件：

- vllm/distributed/parallel_state.py（模块 分布式层；类别 source；类型 core-logic；符号 graph_capture）：修改 graph_capture 函数，新增 graph_capture_context 参数，允许调用者控制捕获流，是实现流选择的关键。
- vllm/v1/worker/gpu_model_runner.py（模块 GPU 模型运行器；类别 source；类型 core-logic；符号 profile_cudagraph_memory）：在 profile_cudagraph_memory() 中添加

ROCm 专用逻辑，使用当前流进行临时分析，这是修复吞吐回归的核心。

- vllm/v1/worker/gpu_worker.py（模块 GPU 工作器；类别 source；类型 core-logic；符号 determine_available_memory）：重新启用 ROCm 的 cudagraph 内存分析入口，将条件从 is_cuda() 改为 is_cuda_alike()。

关键符号：graph_capture, profile_cudagraph_memory, determine_available_memory

关键源码片段

vllm/distributed/parallel_state.py

修改 graph_capture 函数，新增 graph_capture_context 参数，允许调用者控制捕获流，是实现流选择的关键。

```
# vllm/distributed/parallel_state.py

@contextmanager
def graph_capture(
    device: torch.device,
    graph_capture_context: GraphCaptureContext | None = None,
):
    """
    `graph_capture` is a context manager which should surround the code that
    is capturing the CUDA graph. ...

    A caller may pass an explicit ``graph_capture_context`` to control the
    stream used (e.g. to capture on the default stream).
    """
    # 使用传入的上下文，或回退到新建侧流（默认行为）
    context = graph_capture_context or GraphCaptureContext(
        torch.cuda.Stream(device=device)
    )
    with get_tp_group().graph_capture(context), get_pp_group().graph_capture(context):
        yield context
```

vllm/v1/worker/gpu_model_runner.py

在 profile_cudagraph_memory() 中添加 ROCm 专用逻辑，使用当前流进行临时分析，这是修复吞吐回归的核心。

```
# vllm/v1/worker/gpu_model_runner.py (profile_cudagraph_memory 方法内 )

# 在 ROCm 上，捕获这些临时分析图形时使用当前流，
# 而不是 graph_capture() 默认分配的侧流。
# torch 的缓存分配器按流池化空闲块，因此侧流前向传播
# 会将持久化的 aiter MLA sparse 临时缓冲区（约 76 MiB）
# 滞留在单独池中，改变后续真实 KV 缓存的物理布局，
# 导致带宽受限的 decode 内核速度下降约 20%。
# 由于这些图形会被丢弃，此处无需使用侧流。
# cap_ctx=None 在 CUDA 上保持侧流路径，因为 CUDA 的当前流
# 是传统默认流，无法在其上开始捕获。
```

```
cap_ctx = (
    GraphCaptureContext(torch.cuda.current_stream(self.device))
    if current_platform.is_rocm()
    else None
)

try:
    set_cudagraph_capturing_enabled(True)
    with (
        self._freeze_gc(),
        graph_capture(device=self.device, graph_capture_context=cap_ctx),
    ):
        torch.accelerator.synchronize()
        torch.accelerator.empty_cache()
        # 其余分析代码 ...
```

评论区精华

Rohan138提出了一个关键问题：能否在 cudagraph 捕获后删除 AITER 持久化内核缓冲区？peizhang56解释称，该缓冲区（约 76 MiB）是 AITER sparse-MLA *decode* 内核在首次前向传播时延迟分配的临时工作空间，位于 AITER 编译的内核代码内部，没有 Python torch.Tensor 所有权。即使通过 gc.collect() 或 empty_cache() 也无法回收，因为 torch 的缓存分配器按流池化空闲块，而侧流分配导致该缓冲区滞留在非运行流池中。改用当前流捕获后，分配与运行流共享池，因此物理位置与无分析路径一致。Rohan138随后追问为何 get_mla_metadata_v1 不能释放该缓冲区，peizhang56澄清了两种不同的缓冲区集合：元数据缓冲区（torch 可见且持久）与内核内部临时缓冲区（在本问题中起关键作用）。 njhill建议删除 gpu_worker.py 中过时的历史状态注释，peizhang56被要求清理这些注释。

- AITER 持久化缓冲区在侧流分配中的滞留问题 (design): 无法直接删除 AITER 内部缓冲区；改用当前流捕获是可行方案。
- 过时注释清理 (documentation): peizhang56 被要求清理注释，但最终注释已更新而非完全删除。

风险与影响

- 风险：
  1. CUDA 回归风险：PR 中的 ROCm 特定更改（*cap_ctx* 分支）仅在 is_rocm() 为真时生效；CUDA 路径保持不变（*cap_ctx=None*），因此 CUDA 用户的风险较低。但在 CUDA 上测试此更改以确保没有副作用是必要的。
  2. ROCm 特定问题：修复依赖于 ROCm 上 torch.cuda.current_stream() 的行为。如果未来 ROCm 更新改变流语义，该修复可能需要重新评估。
  3. XPU 仍需排除：is_cuda_alike() 包含了 XPU（如 Intel GPU），但 gpu_worker.py 中明确注释 XPU 仍被排除，因为存在未解决的支持问题（参考 #39977）。该逻辑在 gpu_model_runner.py 中没有显式排除 XPU，但 profile_cudagraph_memory 方法仅在 gpu_worker.py 中调用，并且该入口仍被保护。 - 影响：用户影响：在 ROCm 上启用 cudagraph 内存分析，恢复正确内存估计，无需用户干预。基准测试表明吞吐量恢复至

无分析水平（约 83 tok/s）。系统影响：更改了临时图形捕获的内存分配行为，降低了内存碎片化风险。团队影响：简化了代码库，消除了条件性禁用（is_cuda() vsis_cuda_alike()），并为未来贡献者澄清了 ROCm 支持状态。

- 风险标记：核心路径变更，ROCM 特定修复，CUDA 需验证无回归，缺少测试覆盖

关联脉络

- PR #48440 [ROCm] Temporarily disable cudagraph memory profiling: 本 PR 重新启用了该 PR 禁用的功能，并修复了其引入的吞吐回归。
- PR #47366 [Core] Add cudagraph memory profiling: 本 PR 修复了该 PR 在 ROCm 上启用 cudagraph 内存分析后引入的问题。