

PR #55099 完整报告

vllm-project/vllm

[ROCm][Perf][Bugfix] Multi-stream perf improvements; rocprofiler fixes

合并时间: 2026-09-09 08:42

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/55099>

执行摘要

- 一句话: 修复 ROCm 多流性能问题及 rocprofiler 崩溃, 更新 ROCr+CLR 运行时并移除测试变通方案。
- 推荐动作: 本 PR 值得精读, 尤其是对于关注 ROCm 支持、性能优化和 CI/ 稳定性的团队。它不仅仅是一个简单的依赖更新, 而是涉及构建流程优化、上游关键修复集成和测试代码清理的系统性改进。其设计决策 (如构建阶段重排) 反映了对构建系统效率和长期维护的深思熟虑。建议重点关注:
 1. Docker 构建阶段重排背后的推理, 这可以作为优化多阶段 Docker 构建的范例。
 2. 如何系统地追踪和解决由外部库 (ROCm 运行时、rocprofiler) 引起的问题, 并通过上游协作最终移除代码中的变通方案。
 3. 多流性能改进的具体补丁内容 (可通过 PR body 中链接的上游 PR 了解), 这对理解 ROCm 性能调优有参考价值。

功能与动机

动机源于解决多个已报告的问题和性能瓶颈:

- 1) 多流工作负载 (如 #52033, #51794, #51117) 中存在慢跨流依赖, 影响性能;
- 2) DeepEP MoE 测试在 ROCm 上因 rocprofiler-sdk 附着导致进程退出时发生 SIGSEGV 崩溃 (#51644, #51174, #51173);
- 3) 某些分析工作负载仍存在挂起问题 (#54171)。该 PR 通过引入上游 ROCm 系统库的修复来根本解决这些问题。

实现拆解

1. Docker 构建流程优化: 在 docker/Dockerfile.rocm_base 中, 将 ROCr+CLR 运行时的构建从 base 阶段的前置条件移到 components 构建部分之后的一个独立阶段 (build_rocm_runtime)。这样做的原因是: 下游库仅动态链接它们, 此重定位是安全的; 同时, 它避免了 ROCr+CLR 更新时导致所有构建阶段缓存失效并强制重建。
2. 引入上游性能与崩溃修复: 更新 ROCM_RUNTIME_COMMIT 指向包含特定补丁的提交, 这些补丁解决了多流性能问题 (回溯至 ROCm/rocm-systems#11212) 和崩溃问题 (ROCM/rocm-systems#6942)。
3. 移除测试中的临时变通方案: 在 tests/kernels/moe/test_deepest_moe.py 和 tests/kernels/moe/parallel_utils.py 中, 移除了为规避崩溃而添加的 ROC_PROFILER_REGISTER_ENABLED=0 环境变量设置和 os._exit 调用。这些变通方案的

移除表明根本原因已被上游修复解决。

4. 为 ROCm 10.0 镜像应用相同改进：在 docker/Dockerfile.rock_base 中，为 ROCm 10.0 镜像引入了相同的多流性能改进构建步骤。
5. 配置调整：为了解决一个并发构建竞争条件，在 Triton 构建命令中显式设置了 MAX_JOBS=16。

关键文件：

- docker/Dockerfile.rocm_base（模块 构建配置；类别 infra；类型 infrastructure）：核心基础设施文件，包含 ROCm 基础镜像的构建流程重构和运行时库更新，直接影响所有 ROCm 镜像的构建和最终产物。
- docker/Dockerfile.rock_base（模块 构建配置；类别 infra；类型 infrastructure）：为 ROCm 10.0 (TheRock) 镜像引入相同的多流性能改进构建步骤，确保不同 ROCm 版本的基础镜像都包含关键修复。
- tests/kernels/moe/parallel_utils.py（模块 测试工具；类别 test；类型 test-coverage）：移除了为规避 ROCm 崩溃而添加的测试变通方案（os._exit），这标志着根本原因修复，简化了测试代码。
- tests/kernels/moe/test_deepep_moe.py（模块 MoE 测试；类别 test；类型 test-coverage）：移除了测试函数中为 ROCm 平台设置禁用 rocprofiler 的 monkeypatch，依赖上游修复来确保测试正常退出。

关键符号：未识别

关键源码片段

docker/Dockerfile.rocm_base

核心基础设施文件，包含 ROCm 基础镜像的构建流程重构和运行时库更新，直接影响所有 ROCm 镜像的构建和最终产物。

```
###
### ROCR + CLR build
###
# 将此构建阶段从 base 阶段的前置条件移至 components 构建之后。
# 原因：下游库仅动态链接它们，重置此顺序是安全的。
# 主要好处：避免更新 ROCR/CLR 时导致所有下游构建阶段（如 AITER, MORI）缓存失效。
FROM base AS build_rocm_runtime
ARG ROCM_SYSTEMS_REPO
ARG ROCM_RUNTIME_COMMIT
# 安装必要的构建依赖
RUN apt-get update \
    && apt-get install -y --no-install-recommends g++ \
    libelf-dev libdrm-dev libnuma-dev libdw-dev xxd \
    && rm -rf /var/lib/apt/lists/*
RUN pip install --no-cache-dir CppHeaderParser
# 使用 sparse checkout 拉取特定的子项目和提交
RUN git init -q /src && cd /src \
    && git remote add origin ${ROCM_SYSTEMS_REPO} \
```

```

&& git config core.sparseCheckout true \
&& git config remote.origin.promisor true \
&& git config remote.origin.partialclonefilter blob:none \
&& git sparse-checkout init --cone \
&& git sparse-checkout set projects/rocr-runtime projects/clr projects/hip shared cmake \
&& git fetch --filter=blob:none --depth 1 origin ${ROCM_RUNTIME_COMMIT} \
&& git checkout -q FETCH_HEAD
# 构建 ROCr 和 CLR，并安装到临时目录
# ... (省略具体的 cmake 命令) ...

```

docker/Dockerfile.rock_base

为 ROCm 10.0 (TheRock) 镜像引入相同的多流性能改进构建步骤，确保不同 ROCm 版本的基础镜像都包含关键修复。

```

###
### ROCr + CLR build
###
# 引入多流性能改进，在这些改进进入官方发布前使用此构建。
FROM base AS build_rocm_runtime
ARG ROCM_SYSTEMS_REPO
ARG ROCM_RUNTIME_COMMIT
# 安装依赖，包括 libgl-dev 以满足 rocclr 的 OpenGL 查找要求
RUN apt-get update \
    && apt-get install -y --no-install-recommends g++ libdw-dev xxd libgl-dev libglx-dev \
    && rm -rf /var/lib/apt/lists/*
# ... (后续构建步骤类似 Dockerfile.rocm_base) ...

```

评论区精华

核心讨论集中在 Docker 构建阶段的重新组织和 Triton 构建的 **MAX_JOBS** 设置上。

- 构建阶段重排：Rohan138 质疑为何将 ROCr+CLR 构建阶段移到 components 部分之后，而不是放在顶部其他构建阶段之前。mawong-amd 解释说，这是为了避免 ROCr+CLR 更新导致下游昂贵的构建阶段（如 AITER、MORI）缓存失效。尽管 BuildKit 默认不受此影响，但为了兼容非 BuildKit 后端，这是一个务实的权衡。Rohan138 最终接受此安排，并认为未来当修复进入官方 ROCm 发布后，可以更清晰地组织这部分。
- Triton 构建限制：Rohan138 询问是否应该移除 MAX_JOBS=16。mawong-amd 解释这是为了缓解 Triton 构建中一个偶发的并发竞争条件（类似于 triton-lang/triton#6718），并指出 Triton 构建本身很快，其构建时间通常被并行启动的其他长阶段所掩盖，因此这是一个低成本的稳定性改进。Rohan138 表示赞同。
- 变更批准：Rohan138 和 AndreasKaratzas 均批准了 PR。AndreasKaratzas 特别说明，所有 AMD CI 失败都是预先存在的，与本 PR 无关。
 - Docker 构建阶段顺序安排 (design): mawong-amd 的解释基于构建缓存效率的考量被接受。Rohan138 认可这是一个合理的权衡，并指出未来当修复进入官方发布后可以重新调整。
 - Triton 构建 MAX_JOBS 设置 (performance): mawong-amd 的解释被接受，该限制被视为一个低成本的稳定性改进措施。

- ROCm 崩溃修复与测试变通方案移除 (correctness): 问题的根本原因已通过上游修复解决，因此可以安全移除本地变通方案。

风险与影响

- 风险:

1. 兼容性风险: 对 ROCr+CLR 运行时库的更新是核心变更。虽然这些库在下游被动态链接，且此次更新包含重要的错误修复，但任何运行时更改都存在引入新兼容性问题的潜在风险，可能影响依赖这些库的其他组件或用户工作负载。变更涉及两个不同的 Docker 基础镜像 (rocm_base 和 rock_base)，需要确保两者更新的一致性。
2. 测试覆盖: 本 PR 移除了两个测试中的变通方案，这完全依赖于上游修复的有效性。如果上游修复存在漏洞，可能导致这些关键 MoE 测试 (tests/kernels/moe/test_deepep_moe.py) 在 ROCm 上再次失败。
3. 性能影响: 引入的多流性能改进 (回溯至 rocm-systems#11212) 旨在解决特定场景下的性能问题。性能提升的幅度和范围需要通过实际工作负载验证，且需确保不会对单流或特定硬件配置产生负面影响。
4. 构建稳定性: 为 Triton 构建设置 MAX_JOBS=16 是一个预防性措施。虽然旨在解决已知竞争条件，但限制并行度可能略微增加构建时间。

- 影响: 影响范围:

- AMD ROCm 用户: 直接影响使用 vLLM 官方 ROCm Docker 镜像的用户。多流性能改进可能提升他们的工作负载性能，并解决可能导致崩溃的问题。
- vLLM CI/CD: 更新了 CI 使用的 Docker 基础镜像，这直接影响所有 AMD GPU 的测试执行。本 PR 移除的测试变通方案表明，未来 CI 中的这些测试将不再需要特殊处理，运行环境更干净。
- 开发者体验: 对于为 vLLM 开发或调试的工程师，特别是关注 MoE 和分布式测试的开发者，崩溃的修复和变通方案的移除提供了更稳定的开发和测试环境。影响程度:
- 性能: 中等至高。多流性能改进对于使用并发 CUDA/HIP 流的应用程序是显著的。
- 稳定性: 高。解决了进程退出时的崩溃 (SIGSEGV)，提高了使用 vLLM 测试和部署工具的可靠性。
- 维护性: 中等。优化了 Docker 构建流程，减少了因运行时更新导致的缓存失效，有利于更快的镜像迭代。
- 风险标记: 运行时库更新，测试变通方案移除，构建缓存行为变更

关联脉络

- PR #53712 [ROCm] Workaround cross-stream event performance issue: 本 PR 完成了由 #53712 开始的修复系列，后者仅解决了某些流事件的性能问题。本 PR 引入了更全面的上游补丁，解决了跨流依赖的根本问题。
- PR #51174 [ROCm] Work around DeepEP teardown SIGSEGV in MoE test harness: 本 PR 回滚了 #51174 中引入的临时变通方案 (通过 os._exit 退出测试进程)，因为 ROCr+CLR 更新已包含正确的修复。

- PR #51173 [ROCm][CI] Keep rocprofiler-sdk out of DeepEP HT MoE test workers: 本 PR 回滚了 #51173 中引入的临时变通方案（通过设置 ROCPROFILER_REGISTER_ENABLED=0），因为 ROCr+CLR 更新已解决相关崩溃。
- PR #52033 [Bugfix] Fix race condition in distributed executor: 本 PR 解决的多流性能问题与 #52033 中报告的问题相关，这些问题都涉及并发工作负载。