

PR #49925 完整报告

vllm-project/vllm

[ROCm] Add TheRock preview docker updates, Keep Python 3.12 and Ubuntu 22.04

合并时间: 2026-09-01 00:09

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/49925>

执行摘要

- 一句话: ROCm 镜像切换到 TheRock 7.14 构建, 保持 Python 3.12/Ubuntu 22.04
- 推荐动作: 值得精读, 尤其适合维护 CI/ 发布管线的工程师。重点关注三处设计: 一是 Dockerfile.rock 的 REMOTE_VLLM 双模式源码获取与 Rust 分层缓存, 二是 triton constraints 解决 resolver 反复替换版本的工程细节, 三是 TheRock pip 化 SDK 方案的运行时 hack。合并时 RCCL 多 GPU 问题仍未闭环, 建议持续跟踪后续 AMD CI 是否出现 gfx1201 相关回归。

功能与动机

AMD 团队希望让 vLLM 的 ROCm 镜像使用 TheRock 7.14 提供的预编译 wheels, 替代基于 rocm/dev-ubuntu-22.04:7.2.3-complete 的 legacy 构建。评论中 njhill 明确指出 TheRock 包含 legacy ROCm 构建缺失的关键修复 (ROCm/rocm-systems#2451), 这是 extensible kvcache 功能落地的前置条件。PR body 也说明这是 preview 版本, CI 中仍有 MI355 Language Models Tests 与 Entrypoints Integration (Pooling) 两组失败待处理。

实现拆解

实现分四步:

1. 新增 TheRock base 镜像构建 (docker/Dockerfile.rock_base, +360 行): 放弃 rocm/dev-ubuntu-22.04:7.2.3-complete 基础镜像, 改为从 ubuntu:22.04 起步, 通过 pip 安装 `rocm[libraries,devel]==7.14.0` 并执行 `rocm-sdk init` 生成 SDK 布局; 随后按 `PYTORCH_ROCM_ARCH` 逐架构安装 `amd-torch-device-<arch>` 与 `rocm-sdk-device-<arch>` 预编译设备包, 并附带多处 TheRock 运行时修复 (`rtld_global` 改为 `False`、`librocm_smi64.so.7` 符号链接、`ld.so.conf` 写入 SDK 库路径)。
2. 新增完整 vLLM 构建镜像 (docker/Dockerfile.rock, +992 行): 核心是 `REMOTE_VLLM` 双模式源码获取 (本地 COPY 或远端稀疏克隆), 以及 Rust 工具链 / 源码 / 版本探测三个独立阶段以最大化 Docker 层缓存复用; 同时引入 sccache 可选缓存、AITER/MORI/flash-attention 内核构建、Triton 版本 constraints (防止 resolver 把 ROCm 版 triton 换成 CUDA 版 3.7.1) 等。
3. 构建依赖版本对齐 (requirements/build/rocm.txt、requirements/build/rock.txt): index-url 从 <https://download.pytorch.org/whl/rocm7.1> 切换到 <https://repo.amd.com/rocm/whl-multi-arch/>, torch 升到 2.12.0+rocm7.14.0、

torchvision 0.27.0、triton 3.7.1+git0263a6a6、packaging>=26.2；新增 rock.txt 以独立承载 TheRock 依赖链。

4. 测试依赖修复与配置配套 (requirements/test/rocm.in、pyproject.toml)：在 rocm.in 中加入 `bitsandbytes==0.50.0` (TheRock 下 0.49.2 无法正确启用 HIP backend)，并同步更新 rocm.txt lock 文件；pyproject.toml 的 typos 配置补充 `a63ede7` 哈希串白名单。

关键文件：

- docker/Dockerfile.rock_base (模块 镜像构建；类别 infra；类型 infrastructure；符号 base, rocm-sdk init)：新增 TheRock 基础镜像构建文件，是本次迁移的核心：从 ubuntu:22.04 + Python 3.12 起步，通过 pip 安装 ROCm 7.14 SDK 与逐架构设备包，并内嵌多处 TheRock 运行时修复。
- docker/Dockerfile.rock (模块 镜像构建；类别 infra；类型 infrastructure；符号 fetch_vllm, rust-toolchain, vllm-version)：新增完整 vLLM 构建镜像文件，承载 REMOTE_VLLM 双模式源码获取、Rust 分层缓存、sccache、AITER/MORI/FA 内核构建及 triton 约束，是 TheRock 生产构建的入口。
- requirements/test/rocm.in (模块 测试依赖；类别 test；类型 test-coverage)：新增 bitsandbytes==0.50.0 测试依赖，解决 TheRock 下 0.49.2 无法启用 HIP backend 的问题，是 CI 全绿的必要配套。
- requirements/build/rock.txt (模块 构建依赖；类别 docs；类型 documentation)：新增的 TheRock 独立构建依赖清单，明确 torch==2.12.0+rocm7.14.0、triton==3.7.1+git0263a6a6.rocm7.14.0 等关键 wheel 版本。
- requirements/build/rocm.txt (模块 构建依赖；类别 docs；类型 documentation)：原有的 ROCm 构建依赖被整体升级：index-url 切换到 AMD multi-arch 源，torch/torchvision/triton 跨大版本，影响所有 ROCm 构建（含非 TheRock）。
- pyproject.toml (模块 项目配置；类别 config；类型 configuration)：typos 配置新增哈希串 `a63ede7` 白名单，属于配套配置微调。
- requirements/test/rocm.txt (模块 测试依赖；类别 docs；类型 documentation)：rocm.in 对应的 lock 文件同步更新，新增 bitsandbytes==0.50.0 并升级 packaging 到 26.2。

关键符号：未识别

关键源码片段

docker/Dockerfile.rock_base

新增 TheRock 基础镜像构建文件，是本次迁移的核心：从 ubuntu:22.04 + Python 3.12 起步，通过 pip 安装 ROCm 7.14 SDK 与逐架构设备包，并内嵌多处 TheRock 运行时修复。

```
# -----  
# TheRock 方案：不再使用 rocm/dev-ubuntu-22.04:7.2.3-complete 作为 base，  
# 而是从 ubuntu:22.04 起步，用 pip 安装 ROCm 7.14 SDK 的 wheel 包。  
# -----  
ARG SITE_PACKAGES=/usr/local/lib/python${PYTHON_VERSION}/dist-packages  
ARG ROCM_RELEASE_WHEELS_MULTIARCH_URL="https://repo.amd.com/rocm/whl-multi-arch/"
```

```
ARG ROCM_NIGHTLY_WHEELS_MULTIARCH_URL="https://rocm.nightlies.amd.com/whl-multi-arch/"
```

```
ARG ROCM_SDK_VERSION=7.14.0
```

```
# 安装 ROCm SDK: libraries+devel 两个 extra 提供运行时与编译工具链,  
# rocm-sdk init 负责生成统一的 SDK 目录布局 (_rocm_sdk_core / _rocm_sdk_devel) 。  
RUN pip install --index-url ${ROCM_RELEASE_WHEELS_MULTIARCH_URL} \  
    --extra-index-url ${ROCM_NIGHTLY_WHEELS_MULTIARCH_URL} \  
    "rocm[libraries,devel]${ROCM_SDK_VERSION:+==${ROCM_SDK_VERSION}}" \  
    && rocm-sdk init
```

```
# 按 PYTORCH_ROCM_ARCH 逐个安装 AMD 预编译设备包:  
# amd-torch-device-<arch> 让 PyTorch 在对应 GPU 上可直接运行,  
# rocm-sdk-device-<arch> 补齐该架构的设备库; arch 列表覆盖 CDNA 与 RDNA。  
RUN TORCH_VER=$(python3 -c "import torch; print(torch.__version__)") \  
    && for arch in $(echo "${PYTORCH_ROCM_ARCH}" | tr ' ' '); do \  
        echo "Installing device packages for ${arch}" \  
        && pip install --index-url ${ROCM_RELEASE_WHEELS_MULTIARCH_URL} \  
            --extra-index-url ${ROCM_NIGHTLY_WHEELS_MULTIARCH_URL} \  
            amd-torch-device-${arch}==${TORCH_VER} \  
            rocm-sdk-device-${arch}==${ROCM_SDK_VERSION}; \  
    done
```

```
# TheRock 运行时修复:  
# 1) rtd_global 改为 False, 避免 RTLD_GLOBAL 污染符号解析;  
# 2) 为 librocm_smi64 建立 .so.7 符号链接, 兼容既有依赖版本;  
# 3) 把 SDK 库目录写入 ld.so.conf, 让动态链接器能找到 TheRock 的 pip 安装库。  
RUN if [ -f "${SITE_PACKAGES}/rocm_sdk/__init__.py" ]; then \  
    sed -i 's/rtd_global: bool = True/rtd_global: bool = False/g' \  
        "${SITE_PACKAGES}/rocm_sdk/__init__.py"; \  
fi \  
    && if [ -f "${SITE_PACKAGES}/_rocm_sdk_core/lib/librocm_smi64.so.1" ] \  
        && [ ! -e "${ROCM_PATH}/lib/librocm_smi64.so.7" ]; then \  
        ln -sf "${SITE_PACKAGES}/_rocm_sdk_core/lib/librocm_smi64.so.1" \  
            "${ROCM_PATH}/lib/librocm_smi64.so.7"; \  
    fi \  
    && printf '%s\n%s\n' "${ROCM_PATH}/lib" "${SITE_PACKAGES}/_rocm_sdk_core/lib" \  
        > /etc/ld.so.conf.d/rocm-pip.conf \  
    && ldconfig
```

docker/Dockerfile.rock

新增完整 vLLM 构建镜像文件, 承载 REMOTE_VLLM 双模式源码获取、Rust 分层缓存、sccache、AITER/MORI/FA 内核构建及 triton 约束, 是 TheRock 生产构建的入口。

```
# 双模式源码获取: REMOTE_VLLM=0 直接 COPY 本地工作区;  
# REMOTE_VLLM=1 则按 VLLM_REPO/VLLM_BRANCH 稀疏克隆远端仓库。  
# 两种模式最终汇入同一个 fetch_vllm 阶段, 保证下游构建入口一致。  
FROM base AS fetch_vllm_0  
ONBUILD COPY ./vllm/
```

```

FROM base AS fetch_vllm_1
ARG VLLM_REPO
ARG VLLM_BRANCH
ONBUILD RUN git clone ${VLLM_REPO} \
    && cd vllm \
    && git fetch -v --prune -- origin ${VLLM_BRANCH} \
    && git checkout FETCH_HEAD \
    && if [ ${VLLM_REPO} != "https://github.com/vllm-project/vllm.git" ]; then \
        git remote add upstream "https://github.com/vllm-project/vllm.git" \
        && git fetch upstream; \
    fi

```

```

FROM fetch_vllm_${REMOTE_VLLM} AS fetch_vllm

```

```

# Rust 工具链输入单独成阶段：只携带 rust-toolchain.toml,
# 这样 Rust 源码变更不会使携带工具链的 ci_base 长寿命镜像失效。
FROM scratch AS rust_toolchain_input_0
COPY rust-toolchain.toml /rust-toolchain-input/vllm/rust-toolchain.toml

```

```

FROM base AS rust_toolchain_input_1
ARG VLLM_REPO
ARG VLLM_BRANCH
RUN git clone --no-checkout --filter=blob:none ${VLLM_REPO} /rust-toolchain-input/vllm \
    && cd /rust-toolchain-input/vllm \
    && git fetch -v --prune -- origin ${VLLM_BRANCH} \
    && git sparse-checkout init --no-cone \
    && git sparse-checkout set rust-toolchain.toml \
    && git checkout FETCH_HEAD

```

```

FROM rust_toolchain_input_${REMOTE_VLLM} AS rust-toolchain-input

```

评论区精华

1. AITER 内核预编译开关：tjtanaa 质疑 AITER_PREBUILD_AITER_KERNELS=0 会让 AITER 在运行时 JIT 编译 C++ 代码导致极慢，AndreasKaratzas 表示 100% 同意；rasmith 解释 draft 阶段仅为开发便利，随后直接移除了该选项。
2. bitsandbytes 兼容性：AndreasKaratzas 追问为何必须绕过 PyPI 指到 rasmith 的 fork，rasmith 说明一是被 bitsandbytes-foundation/bitsandbytes#1889 修复（影响 TheRock），二是 HIP backend 检测失败只有 CPU backend；最终切换为官方 bitsandbytes==0.50.0。
3. Triton 版本 resolver 陷阱：rasmith 自述在 Dockerfile.rock 中至少有 5 处会被 resolver 把 ROCm 版 triton 换成 3.7.1 或降级到 CUDA 版 2.13，必须用 constraints 文件锁定，且源码安装缺 METADATA/RECORD 会导致无法阻止的 reinstall，需要 in-place 补丁。
4. RCCL 多 GPU 启动失败（未闭环）：tanpinsiang 报告在 gfx1201（Radeon AI PRO R9700）上 TP4/TP8 均报 "HIP failure: invalid device pointer"，无模型参与的 torch.distributed all-reduce 即可复现，而 nightly 与 rocm7.14.0 dev 镜像正常；

rasmith 请求提供 reproducer。合并者 dllehr-amd 的结论是："If we can recreate the rccl error lets tackle it. Otherwise lets go"。

- 5. MORI 版本调整：Rohan138 询问为什么 pin 到 f95cd5e, rasmith 解释该 commit 修复了 TheRock 下 MORI CMake 找不到 numa 的问题，并提到 ROCm/mori#541。
- AITER 内核是否应在构建期预编译 (design): 移除预编译开关，回归 AITER 构建期预编译默认行为。
- bitsandbytes 版本与 TheRock 兼容性 (correctness): 采用官方 bitsandbytes==0.50.0, 放弃 fork 依赖。
- Triton 版本被 resolver 反复覆盖 (design): 保留 constraints 方案，并吸收 PR#52819 中的 pybind 与 triton pin。
- RCCL 多 GPU 启动失败 (gfx1201) (correctness): 合并时未复现、未定位根因，作为已知问题遗留。
- MORI 版本 pin 的缘由 (question): 维持 f95cd5e pin, 等待 MORI 后续版本统一支持 TheRock。

风险与影响

- 风险:
 1. RCCL 启动失败未定位根因 (gfx1201 多卡场景) : TP4/TP8 均报 HIP failure: invalid device pointer 且与模型无关，合并时未复现、未闭环，可能影响新 RDNA 架构 (gfx1201) 的多卡用户。
 2. 全新镜像构建链路: Dockerfile.rock/rock_base 为从零新增，构建时间、缓存命中率、镜像体积均与旧链路不同；rust-input/toolchain 分层设计依赖 Docker 多阶段缓存行为，远端构建可能存在不确定性。
 3. 依赖跨大版本升级: torch 2.11 -> 2.12+rocm7.14.0、triton 3.6 -> 3.7.1 会改变 kernel 行为与算子实现；requirements/build/rocm.txt 同时影响非 TheRock 的 ROCm 构建，风险面不止新镜像。
 4. TheRock 运行时 hack 的脆弱性: sed 修改 rocm_sdk/__init__.py 的 rtdl_global、手工建立 librocm_smi64.so.7 符号链接，都属于对上游 wheel 的 in-place 修改，上游一旦变更布局这些 hack 会静默失效。- 影响: 影响范围是整个 AMD ROCm 用户群体与发布管线: 合并后 vLLM 的 ROCm 镜像 /CI 将以 TheRock 7.14 + torch 2.12 为基线，与 legacy ROCm 7.2.3 形成双轨 (Rohan138 明确表示会同时发布两个版本一段时间)。对团队的直接影响是 AMD CI 全部改用新镜像，构建脚本维护成本转移；对用户的间接影响是后续 ROCm nightly 容器会携带 TheRock 修复 (如 rocm-systems#2451)，这是 extensible kvcache 等新功能在 AMD 上落地的前置条件。Python 3.12 与 Ubuntu 22.04 保持不变，兼容面受控。- 风险标记: RCCL 启动失败未闭环，全新镜像构建链路，跨大版本依赖升级，TheRock 运行时 hack 脆弱

关联脉络

- PR #52819 pybind pin 3.0.4 与 triton commit 对齐 (讨论中引用，确切标题未知) : Rohan138 在 review 中要求本 PR 同步 52819 的 pybind 3.0.4 pin 与新 triton commit,

且 52819 会先合并。

- PR #52826 AITER 版本 bump 到 0.1.20 (讨论中引用, 确切标题未知): Rohan138 表示将两个 Dockerfile 的 AITER 统一升级到 0.1.20, 作为本 PR 的后续对齐。
- PR #51721 ROCM_TRITON_KERNELS_COMMIT 固定来源 (讨论中引用, 确切标题未知): rasmith 在回复 Rohan138 关于 triton kernels pin 时指出该固定值来自 PR#51721。