PR #46527 完整报告

vllm-project/vllm

[ROCm][CI] Cache Rust builds by source inputs

合并时间：2026-07-13 14:14

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/46527

执行摘要

- 一句话：ROCM CI Rust 构建缓存优化，避免非 Rust 变更时重复编译
- 推荐动作：推荐 CI 和构建团队精读，特别是 Docker 多阶段缓存和内容哈希模式，可复用于其他高频编译模块。

功能与动机

PR body 指出 ROCm Docker 路径中即使 PR 未改动 Rust 代码，Rust 构建也会被触发，因为 build_rixl 阶段继承了完整的源代码树，任何仓库变更都会导致 cargo 构建层失效。通过缩小输入范围并引入内容哈希缓存，避免不必要的重复编译。

实现拆解

1. 分离 Rust 工具链与源码输入阶段（docker/Dockerfile.rocm）：新增 rust_toolchain_input_0/1 和 rust_input_0/1 阶段，分别使用 scratch 拷贝或 git sparse-checkout 仅获取必需的 Rust 文件，最终通过 REMOTE_VLLM 参数选择本地或远程源。
2. 添加内容哈希计算与缓存参数（.buildkite/scripts/ci-bake-rocm.sh）：新增 compute_rocm_rust_content_hash 和 compute_rocm_rust_content_hash_if_needed 函数，计算 Rust 输入文件的哈希值，并设置 --cache-from 和 --cache-to 参数将哈希写入 Docker 缓存标签。
3. 配置 HCL 缓存函数（docker/ci-rocm.hcl）：新增 get_cache_from_rocm_rust 和 get_cache_to_rocm_rust 函数，以及 ROCM_RUST_CACHE_TO_MODE 变量，使 Rust 构建层的缓存可独立读取和导出。

配套修改：将 rust-toolchain.toml 等文件加入 CI 基础镜像内容文件列表，确保哈希准确。

关键文件：

- .buildkite/scripts/ci-bake-rocm.sh（模块 CI 脚本；类别 infra；类型 core-logic；符号 uses_rocm_rust_cache, compute_rocm_rust_content_hash, compute_rocm_rust_content_hash_if_needed, compute_rocm_rust_cache_ref）：包含核心逻辑：Rust 内容哈希计算、缓存参数注入、目标判断，是缓存生效的控制中心。
- docker/Dockerfile.rocm（模块 Dockerfile；类别 infra；类型 infrastructure；符号 rust_toolchain_input_0, rust_toolchain_input_1, rust-toolchain-input, rust_input_0）：Docker 构建定义，重组 Rust 阶段以分离输入，实现精确缓存依赖。

- docker/ci-rocm.hcl（模块 构建配置；类别 infra；类型 infrastructure；符号 *get_cache_from_rocm_rust*, *get_cache_to_rocm_rust*, ROCM_RUST_CACHE_TO_MODE）：HCL 配置文件，定义 Rust 缓存导入 / 导出函数，与 Shell 脚本配合生成正确的 *--cache-from/to* 参数。

关键符号：*uses_rocm_rust_cache*, *compute_rocm_rust_content_hash*, *compute_rocm_rust_content_hash_if_needed*, *get_cache_from_rocm_rust*, *get_cache_to_rocm_rust*

关键源码片段

docker/Dockerfile.rocm

Docker 构建定义，重组 Rust 阶段以分离输入，实现精确缓存依赖。

```
# Rust/protoc toolchain inputs——独立于源码，避免源码变更影响工具链缓存
FROM scratch AS rust_toolchain_input_0
COPY rust-toolchain.toml /rust-toolchain-input/vllm/rust-toolchain.toml
COPY tools/install_protoc.sh /rust-toolchain-input/vllm/tools/install_protoc.sh

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 \
       tools/install_protoc.sh \
    && git checkout FETCH_HEAD

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

# Rust build inputs——仅包含编译 Rust 前端所需的文件
FROM scratch AS rust_input_0
COPY requirements/build/rust.txt /rust-input/vllm/requirements/build/rust.txt
COPY rust/Cargo.lock rust/Cargo.toml /rust-input/vllm/rust/
COPY rust/proto /rust-input/vllm/rust/proto
COPY rust/src /rust-input/vllm/rust/src
COPY rust-toolchain.toml /rust-input/vllm/rust-toolchain.toml
COPY tools/build_rust.py /rust-input/vllm/tools/build_rust.py
COPY build_rust.sh /rust-input/vllm/build_rust.sh

FROM base AS rust_input_1
ARG VLLM_REPO
ARG VLLM_BRANCH
RUN git clone --no-checkout --filter=blob:none ${VLLM_REPO} /rust-input/vllm \
    && cd /rust-input/vllm \
    && git fetch -v --prune -- origin ${VLLM_BRANCH} \
```

```
    && git sparse-checkout init --no-cone \
    && git sparse-checkout set \
      requirements/build/rust.txt \
      rust/Cargo.lock \
      rust/Cargo.toml \
      rust/proto/** \
      rust/src/** \
      rust-toolchain.toml \
      tools/build_rust.py \
      build_rust.sh \
    && git checkout FETCH_HEAD

FROM rust_input_${REMOTE_VLLM} AS rust-input
```

docker/ci-rocm.hcl

HCL 配置文件，定义 Rust 缓存导入 / 导出函数，与 Shell 脚本配合生成正确的 --cache-from/to 参数。

```
# 从注册表拉取 Rust 构建缓存（精确提交、父提交、合并基、分支标签）
function "get_cache_from_rocm_rust" {
  params = []
  result = compact([
    BUILDKITE_COMMIT != "" ? "type=registry,ref=${DOCKERHUB_CACHE_REPO}:rust-rocm-${BUILDKITE_COMMIT}" : "",
    PARENT_COMMIT != "" ? "type=registry,ref=${DOCKERHUB_CACHE_REPO}:rust-rocm-${PARENT_COMMIT}" : "",
    VLLM_MERGE_BASE_COMMIT != "" ? "type=registry,ref=${DOCKERHUB_CACHE_REPO}:rust-rocm-${VLLM_MERGE_BASE_COMMIT}" : "",
    ROCM_CACHE_BRANCH_TAG != "" ? "type=registry,ref=${DOCKERHUB_CACHE_REPO}:rust-rocm-branch-${ROCM_CACHE_BRANCH_TAG}" : "",
    ROCM_CACHE_UPSTREAM_BRANCH_TAG != "" ? "type=registry,ref=${DOCKERHUB_CACHE_REPO}:rust-rocm-branch-${ROCM_CACHE_UPSTREAM_BRANCH_TAG}" : "",
  ])
}

# 导出 Rust 构建缓存到注册表（精确提交和分支标签，模式由变量控制）
function "get_cache_to_rocm_rust" {
  params = []
  result = compact([
    BUILDKITE_COMMIT != "" ? "type=registry,ref=${DOCKERHUB_CACHE_REPO}:rust-rocm-${BUILDKITE_COMMIT},mode=${ROCM_RUST_CACHE_TO_MODE}" : "",
    ROCM_CACHE_BRANCH_TAG != "" ? "type=registry,ref=${DOCKERHUB_CACHE_REPO}:rust-rocm-branch-${ROCM_CACHE_BRANCH_TAG},mode=${ROCM_RUST_CACHE_TO_MODE}" : "",
  ])
}
```

评论区精华

- @tjtanaa 询问 AMD CI 是否使用 REMOTE_VLLM=1，作者确认使用但指出历史不清晰。该参数控制从远程仓库克隆代码，影响缓存输入路径。
- @depthfirst-app[bot] 建议对 curl ... | sh 远程脚本安装增加校验和验证，降低安全风险。作者未回应此建议。
- AMD CI 是否使用 REMOTE_VLLM=1 (question): AndreasKaratzas 确认 AMD CI 确实使用 REMOTE_VLLM=1，但指出历史不明确。
- 远程安装脚本安全风险 (security): 未在 PR 中解决，但严重性标记为 LOW。

风险与影响

- 风险：
  - 缓存逻辑复杂，新增多个阶段和哈希计算，可能引入缓存未命中或构建失败的风险，尤其是 REMOTE_VLLM=1 的网络依赖。
  - 安全建议未采纳：rustup 安装脚本直接管道执行，理论上存在被篡改风险，但 CI 环境通常可控。
  - 影响范围仅限 ROCm 后端 CI，其他平台无影响。
- 影响：
  - 用户：降低 ROCm Docker 镜像构建时间，减少 CI 资源消耗。
  - 系统：通过内容哈希缓存避免全量 Rust 重编译，提升构建效率。
  - 团队：后续维护需注意 Dockerfile.rocm 中阶段命名和缓存标签的兼容性。
  - 风险标记：远程脚本无校验和，REMOTE_VLLM 依赖网络稳定性，缓存阶段命名硬编码

关联脉络

- PR #48222 [CI][Rust Frontend] Pin cargo tool versions: 同属 Rust 相关 CI 基础设施改进，该 PR 固定工具版本，此 PR 优化构建缓存。