PR #39721 完整报告

vllm-project/vllm

[ROCm] ROCm DeepEP API updated to latest

合并时间：2026-04-30 22:47

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/39721

执行摘要

- 一句话：统一 ROCm DeepEP API 并更新 Docker 构建
- 推荐动作：该 PR 值得精读，尤其是关注如何通过移除平台分支实现 API 统一，以及 Docker 构建的最佳实践（如条件安装 rdma-core）。对于 ROCm 平台开发者有重要参考价值。

功能与动机

PR 描述指出目的是“Enablement of all the DeepEP API parameters”，使 ROCm DeepEP API 与 vLLM 匹配，支持低延迟等参数，并更新依赖版本。

实现拆解

实现按以下步骤：

1. 移除平台条件分支：在 deepep_ll.py 中删除了 current_platform.is_rocm() 分支，使 ROCm 和 NVIDIA 使用相同的 low_latency_dispatch 调用，统一传入了 round_scale、use_ue8m0、use_nvfp4 等参数。
2. 统一通信管理器参数：在 all2all.py 中移除 current_platform.is_rocm() 条件，为 DeepEPHTAll2AllManager 和 DeepEPLLAll2AllManager 的 _make_all2all_kwargs 方法直接添加 explicitly_destroy=True、allow_nvlink_for_low_latency_mode=True 等参数，消除 ROCm 与 NVIDIA 差异。
3. 更新 Docker 构建：升级 ROCSHMEM 和 DeepEP 仓库 commit；改用 all_backends 脚本构建 ROCSHMEM；为 DeepEP 构建添加 --rocm-explicit-ctxflag；根据 DEEPEP_NIC 变量（cx7 或 io）条件安装 rdma-core v62 以支持最新 ROCSHMEM。

关键文件：

- vllm/model_executor/layers/fused_moe/prepare_finalize/deepep_ll.py（模块 MoE 调度；类别 source；类型 data-contract；符号 prepare_async）：核心 MoE dispatch 逻辑，移除平台条件分支，统一参数传递。
- vllm/distributed/device_communicators/all2all.py（模块 通信层；类别 source；类型 dependency-wiring；符号 DeepEPHTAll2AllManager._make_all2all_kwargs, DeepEPLLAll2AllManager._make_all2all_kwargs）：DeepEP 通信管理器，移除 ROCm 条件，统一参数构造。

- docker/Dockerfile.rocm（模块 部署脚本；类别 infra；类型 infrastructure）：更新 ROCSHMEM 和 DeepEP 版本，调整构建方式，安装 rdma-core。

关键符号：prepare_async, _make_all2all_kwargs, get_handle

关键源码片段

vllm/model_executor/layers/fused_moe/prepare_finalize/deepep_ll.py

核心 MoE dispatch 逻辑，移除平台条件分支，统一参数传递。

```
# 变更后统一调用（ROCm 与 NVIDIA 相同路径）
expert_x, expert_num_tokens, handle, _, hook = self.buffer.low_latency_dispatch(
    a1, dispatch_topk_ids, self.max_tokens_per_rank, num_experts,
    use_fp8=self.use_fp8_dispatch,
    round_scale=self.use_ue8m0_dispatch,
    use_ue8m0=self.use_ue8m0_dispatch,
    **(dict(use_nvfp4=True) if use_nvfp4 else dict()),
    **(dict(x_global_scale=qc_a1_gscale_or_scale) if qc_a1_gscale_or_scale is not None and
    nvfp4_dispatch else dict()),
    async_finish=False,
    return_recv_hook=True,
)
```

docker/Dockerfile.rocm

更新 ROCSHMEM 和 DeepEP 版本，调整构建方式，安装 rdma-core。

```
# 更新 commit ID
ARG ROCSHMEM_BRANCH=f0acb0c6
ARG DEEPEP_BRANCH=a9ea9774
# 使用推荐构建脚本（替换显式 cmake）
RUN INSTALL_PREFIX=${ROCSHMEM_DIR} ../scripts/build_configs/all_backends -DUSE_
EXTERNAL_MPI=OFF
# 为 DeepEP 构建添加 --rocm-explicit-ctx
RUN python3 setup.py --variant rocm --rocm-explicit-ctx --nic ${DEEPEP_NIC} bdist_wheel
# 条件安装 rdma-core（仅 cx7/io）
RUN if [ '${DEEPEP_NIC}' = 'cx7' ] || [ '${DEEPEP_NIC}' = 'io' ]; then git clone --branch v62.0 -
-depth 1 https://github.com/linux-rdma/rdma-core.git /tmp/rdma-core && cd /tmp/rdma-core &&
mkdir -p build && cd build && cmake -GNinja -DCMAKE_INSTALL_PREFIX=/usr -DNO_MAN_
PAGES=1 .. && ninja && ninja install && ldconfig && rm -rf /tmp/rdma-core; fi
```

评论区精华

Review 中主要讨论包括：

- rdma-core 安装阶段：gemini-code-assist 建议将 rdma-core 安装移到 base stage 以避免 ABI 不兼容。作者回应 DeepEP 仅用于 test stage，rdma-core 是运行时依赖，在 test stage 安装已验证通过。
- ROCSHMEM 构建脚本：gemini-code-assist 质疑 all_backends 脚本是否正确处理安装和 INSTALL_PREFIX。作者回应该脚本是 ROCSHMEM 官方推荐方法，适用于所有 NIC 类型。

- rdma-core 安装阶段 (design): 作者回应 DeepEP 仅在 test stage 中使用（不出现在 final stage），rdma-core 是运行时依赖，在 test stage 安装已验证通过。
- ROCSHMEM 构建脚本变更 (design): 作者说明 all_backends 是 ROCSHMEM 官方推荐的构建方法，适用于所有 NIC 类型。

风险与影响

- 风险：主要风险包括：
  - Docker 构建一致性：rdma-core 仅在 test stage 安装，若后续将 DeepEP 用于其他 stage 可能缺少依赖。目前作者确认仅在 test stage 使用。
  - 统一调用路径：移除平台分支后，ROCm 使用完整的参数集（如 use_nvfp4），需要底层 DeepEP 库支持。若 ROCm DeepEP 版本不匹配，可能引发运行时错误。已验证 GSM8K 精度一致。
  - 构建脚本替换：从显式 cmake/make 切换到 all_backends 脚本，若脚本行为有差异可能影响 ROCSHMEM 安装。作者表示是官方推荐方法。
- 影响：影响范围：
  - 用户：使用 ROCm 和 DeepEP 的用户现在可以无缝使用低延迟和高速率模式，无需手动设置环境变量。也支持了新 NIC 类型（Thor2）。
  - 系统：统一平台分支减少了未来维护成本，但需要确保 ROCm 后端持续同步更新。
  - 团队：简化了 ROCm 和 NVIDIA 之间的代码分歧，有利于后续 DeepEP 相关开发。
  - 风险标记：Docker 构建阶段依赖，统一平台分支潜在回归

关联脉络

- 暂无明显关联 PR