PR #28348 完整报告

sgl-project/sglang

[AMD]: Enable NIXL PD disaggregation for ROCm(1/n)

合并时间：2026-06-30 12:23

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/28348

执行摘要

- 一句话：ROCm Docker 镜像集成 NIXL PD 传输后端
- 推荐动作：值得精读，尤其关注 Dockerfile 中构建隔离问题的处理（--no-build-isolation、--no-deps）、taskflow pkgconfig 的 workaround，以及 CI 验证步骤的设计。这些工程经验可推广到其他第三方库的集成。

功能与动机

PD（prefill/decode）disaggregation 可以通过 NIXL 传输 KV cache，但 ROCm 镜像没有 NIXL 构建，导致 --disaggregation-transfer-backend nixl 在 AMD GPU 上不可用。PR 旨在提供可选的、可复现的 NIXL 构建，使 AMD 用户也能使用 NIXL 后端。

实现拆解

1. Dockerfile 构建阶段：在 docker/rocm.Dockerfile 中新增可选阶段（通过 --build-arg ENABLE_NIXL=1 控制），安装构建依赖（autoconf、rdma-core 等），从源码构建 UCX（--with-rocm）和 ai-dynamo/nixl（固定 commit c28061f），并使用 --no-build-isolation 复用镜像中的 ROCm torch，避免拉取 CUDA torch。最终创建 nixl 到 nixl_rocm 的符号链接。
2. CI 验证步骤：在 pr-test-amd.yml 和 pr-test-amd-rocm720.yml 中添加"Verify NIXL in Container"步骤，测试 import nixl、UCX 库存在、ucx_info 报告 ROCm 支持、以及 NIXL agent 初始化。
3. 端到端测试：新增 test/registered/amd/disaggregation/test_nixl_transfer_engine_e2e.py，注册到 stage-b-test-large-8-gpu-mi35x-disaggregation-amd 套件。测试类 NixlTransferEngineBase 继承 PDDisaggregationServerBase，自动跳过无 NIXL 或不满足 GPU 数量的环境，启动预填充与解码服务器并运行 GSM8K 评估。
4. 夜间构建：在 release-docker-amd-nightly.yml 和 release-docker-amd-rocm720-nightly.yml 中添加 --build-arg ENABLE_NIXL=1，确保每日镜像包含 NIXL。

关键文件：

- test/registered/amd/disaggregation/test_nixl_transfer_engine_e2e.py（模块 测试；类别 test；类型 test-coverage；符号 NixlTransferEngineBase, setUpClass, tearDownClass, _shift_ports）：新增的端到端测试，覆盖从镜像验证到模型推理的全流程

，是 NIXL 功能在 ROCm 上的质量保障。

- docker/rocm.Dockerfile（模块 部署镜像；类别 infra；类型 infrastructure）：核心构建文件，添加了 UCX 和 NIXL 的构建步骤，并通过构建参数控制，是功能启用的基础。
- .github/workflows/pr-test-amd.yml（模块 CI 配置；类别 infra；类型 infrastructure）：在 AMD CI 中添加 NIXL 验证步骤，确保镜像中 NIXL 可用。
- .github/workflows/pr-test-amd-rocm720.yml（模块 CI 配置；类别 infra；类型 infrastructure）：与 pr-test-amd.yml 相同，针对 ROCm 7.2 的 CI 配置。
- .github/workflows/release-docker-amd-nightly.yml（模块 CI 配置；类别 infra；类型 infrastructure）：夜间构建添加 ENABLE_NIXL=1，确保每日镜像包含 NIXL。
- .github/workflows/release-docker-amd-rocm720-nightly.yml（模块 CI 配置；类别 infra；类型 infrastructure）：与 release-docker-amd-nightly.yml 相同，针对 ROCm 7.2 的夜间构建。

关键符号：NixlTransferEngineBase.setUpClass, NixlTransferEngineBase.tearDownClass, NixlTransferEngineBase._shift_ports, NixlTransferEngineBase.start_prefill, NixlTransferEngineBase.start_decode

关键源码片段

test/registered/amd/disaggregation/test_nixl_transfer_engine_e2e.py

新增的端到端测试，覆盖从镜像验证到模型推理的全流程，是 NIXL 功能在 ROCm 上的质量保障。

```
import os
import unittest
from sglang.test.ci.ci_register import register_amd_ci
from sglang.test.few_shot_gsm8k import run_eval as run_eval_few_shot_gsm8k
from sglang.test.server_fixtures.disaggregation_fixture import PDDisaggregationServerBase
from sglang.test.test_utils import (
    DEFAULT_SMALL_MODEL_NAME_FOR_TEST,
    popen_launch_pd_server,
    try_cached_model,
)

register_amd_ci(est_time=900, suite="stage-b-test-large-8-gpu-mi35x-disaggregation-amd")


class NixlTransferEngineBase(PDDisaggregationServerBase):
    """PD-over-NIXL e2e on ROCm. NIXL (upstream ai-dynamo/nixl + UCX --with-rocm)
    is enabled by default in the ROCm image; when the image was built with
    `--build-arg ENABLE_NIXL=0`, `import nixl` fails and the test skips rather
    than failing the suite."""

    required_gpus = 2
    model_default = DEFAULT_SMALL_MODEL_NAME_FOR_TEST

    @classmethod
```

```
    def setUpClass(cls):
        # 跳过 CUDA 不可用或 GPU 不足的环境
        try:
            import torch
            if not torch.cuda.is_available():
                raise unittest.SkipTest("torch.cuda is not available.")
            if torch.cuda.device_count() < cls.required_gpus:
                raise unittest.SkipTest(
                    f"NIXL PD smoke test requires >= {cls.required_gpus} visible GPUs."
                )
        except unittest.SkipTest:
            raise
        except Exception as e:
            raise unittest.SkipTest(f"torch is not available/usable: {e}")

        # 跳过未安装 NIXL 的环境
        try:
            import nixl # noqa: F401
        except Exception as e:
            raise unittest.SkipTest(
                "nixl not importable; image may have been built with "
                "--build-arg ENABLE_NIXL=0 "
                f"({e})."
            )

        super().setUpClass()
        cls.transfer_backend = ["--disaggregation-transfer-backend", "nixl"]
        # 可选 RDMA 配置
        rdma_env = os.environ.get("SGLANG_TEST_RDMA_DEVICE")
        cls.rdma_devices = ["--disaggregation-ib-device", rdma_env] if rdma_env else []

        cls.model = try_cached_model(
            os.environ.get(cls.model_env_var, cls.model_default)
        )
        cls.start_prefill()
        cls.start_decode()
        cls.wait_server_ready(cls.prefill_url + "/health", timeout=300, process=cls.process_prefill)
        cls.wait_server_ready(cls.decode_url + "/health", timeout=300, process=cls.process_
        decode)
        cls.launch_lb()
```

docker/rocm.Dockerfile

核心构建文件，添加了 UCX 和 NIXL 的构建步骤，并通过构建参数控制，是功能启用的基础。

```
# 构建参数控制是否启用 NIXL（默认启用）
ARG ENABLE_NIXL=1
ARG UCX_REPO="https://github.com/openucx/ucx.git"
ARG UCX_BRANCH="v1.19.x"
ARG NIXL_REPO="https://github.com/ai-dynamo/nixl.git"
```

```
ARG NIXL_COMMIT="c28061f9782e099f975bcc79198b7b5a1a36cc40"

# 构建 NIXL（ENABLE_NIXL=1 时执行）
RUN /bin/bash -lc 'set -euo pipefail; \
    [ "${ENABLE_NIXL}" = "1" ] || { echo "[NIXL] skip"; exit 0; }; \
    apt-get update && apt-get install -y --no-install-recommends \
        build-essential autoconf automake libtool pkg-config git \
        libibverbs-dev librdmacm-dev rdma-core && rm -rf /var/lib/apt/lists/*; \
    pip install --no-cache-dir meson ninja pybind11 meson-python patchelf pyyaml; \
    # 构建 UCX（--with-rocm）
    git clone --depth=1 -b "${UCX_BRANCH}" "${UCX_REPO}" /sgl-workspace/ucx; \
    cd /sgl-workspace/ucx && ./autogen.sh && ./configure --prefix=/opt/ucx \
        --with-rocm=/opt/rocm --with-verbs --with-dm --enable-mt && make -j$(nproc) && make
        install; \
    # 构建 nixl（--no-build-isolation 避免拉取 CUDA torch）
    git clone --depth=1 "${NIXL_REPO}" /sgl-workspace/nixl; \
    cd /sgl-workspace/nixl && git checkout "${NIXL_COMMIT}"; \
    meson setup build --prefix=/usr/local -Ducx_path=/opt/ucx \
        -Dwheel_variant=rocm -Denable_plugins=UCX,POSIX; \
    ninja -C build install; \
    # SGLang 导入 "nixl"，所以需要符号链接
    ln -sf /usr/local/lib/python*/site-packages/nixl_rocm* /usr/local/lib/python*/site-packages/nixl;
    \
    echo "export LD_LIBRARY_PATH=/opt/ucx/lib:\${LD_LIBRARY_PATH}" >> /etc/bash.bashrc; \
    echo "[NIXL] Done."'
```

评论区精华

主要讨论围绕 CI 验证与测试覆盖。@bingxche 在多个 ROCm 版本（7.0、7.2）和不同 GPU（MI30x、MI35x）上验证了镜像构建和 NIXL 功能，测试均通过。@bingxche 要求将 NIXL 测试注册到 CI 套件中，并确认在 stage-b-disagg 和 moriep_small 场景下通过。最终由 @HaiShaw 批准合并。

- CI 验证与测试覆盖 (testing): 所有测试通过，CI 验证步骤通过，端到端测试注册到 stage-b-disagg 套件。

风险与影响

- 风险：
  1. 外部依赖构建：UCX 和 NIXL 从源码构建，依赖网络可用性和上游仓库稳定性；若上游 commit 被强制推送或 GitHub 归档变更，可能导致构建失败。
  2. 构建时间增加：默认启用 NIXL 会增加 Docker 构建时间（约 5-10 分钟）。
  3. 镜像大小：引入 UCX 库和 NIXL 包，可能增加几百 MB。
  4. 测试覆盖不足：端到端测试仅覆盖 small model 和 GSM8K，未验证大规模模型或高并发场景。
  5. API 兼容性：NIXL 和 UCX 版本锁定，若未来 SGLang 更新 NIXL API 可能不兼容。 - 影响：对用户：AMD ROCm 用户现在可以使用 --disaggregation-transfer-backend

nixl 进行 PD 拆分，获得与 CUDA 相同的功能。对系统：默认 Docker 镜像将包含NIXL 相关库，但旧镜像不变，向后兼容。对团队：CI 增加 NIXL 验证步骤，确保每次ROCm 镜像变更不会破坏 NIXL 功能。维护者需关注上游 NIXL 和 UCX 的更新。

- 风险标记：外部依赖构建，构建时间增加，上游提交锁定，测试覆盖有限

关联脉络

- PR #29671 [AMD] Register fused_metadata_copy JIT kernel test for AMD nightly CI: 同为 AMD CI 相关，涉及测试注册和镜像验证。
- PR #29499 [DSA] Optimize DSA CUDA graph replay metadata generation: 此前 PR 也涉及 PD disaggregation 的优化，但针对 CUDA，本 PR 将其扩展到 ROCm。