PR #46573 完整报告

vllm-project/vllm

[ROCm][CI] Expand basic correctness target suites

合并时间：2026-06-25 12:18

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/46573

执行摘要

- 一句话：ROCm CI 基本正确性测试套件自动检测硬件并支持每设备内存阈值
- 推荐动作：本 PR 虽然是测试基础设施改动，但包含了良好的工程实践：向后兼容的环境变量升级、自动硬件检测、可复用 GPU 内存工具。值得精读其设计思路，特别是如何在不破坏现有 CI 配置的前提下逐步迁移。

功能与动机

PR body 说明：此 PR 从 #39238 中拆分出 basic-correctness target-suite 变更，更新分布式 basic-correctness 参数化，使 ROCm 套件能够运行预期的双 GPU 用例，而不是被限制在 L4/A100。

实现拆解

1. 自动目标套件检测：在 tests/basic_correctness/test_basic_correctness.py 中新增 _default_target_test_suite() 函数，通过 current_platform.get_device_name() 和 ROCm 平台特定函数（on_gfx950、on_gfx942）识别 MI250/MI300/MI325/MI355，非 ROCm 默认返回 L4。_resolve_target_test_suite() 优先读取环境变量 VLLM_TARGET_TEST_SUITE 和兼容旧的 TARGET_TEST_SUITE，否则使用默认值。
2. 每设备内存阈值：在 tests/utils.py 中重构 wait_for_gpu_memory_to_clear，将其 threshold_bytes 和 threshold_ratio 参数类型从单一 int/float 扩展为 int/floatldict[int, int/float]，允许对每个 GPU 设备独立设置阈值。提取 record_gpu_memory_usage_stats 函数复用 GPU 内存查询逻辑，并添加 get_physical_device_indices 类型注解。
3. HfRunner 启动内存记录：在 tests/conftest.py 中修改 HfRunner.__enter__，在 ROCm 平台下调用 record_gpu_memory_usage_stats 记录当前内存占用，计算阈值比例（0.05 + 当前已用比例）存入 self.threshold_ratios。__exit__ 将阈值传入 wait_for_rocm_memory_to_settle 并在退出后清理该属性。
4. CI 配置更新：在 .buildkite/test-amd.yaml 中更新 MI250、MI300、MI325 队列的命令，将 TARGET_TEST_SUITE 从 L4/A100 改为对应的硬件名，例如 MI250 队列使用 MI250，MI300 队列使用 MI300，MI325 队列也使用 MI300（因为 MI325 与 MI300 共用部分用例）。

关键文件：

- tests/utils.py（模块 测试工具；类别 test；类型 test-coverage；符号 get_physical_device_indices, record_gpu_memory_usage_stats）：重构 wait_for_gpu_memory_to_clear 支持 per-device 阈值，新增 record_gpu_memory_usage_stats 辅助函数，并为 get_physical_device_indices 添加类型注解。
- tests/basic_correctness/test_basic_correctness.py（模块 正确性测试；类别 test；类型 test-coverage；符号 _default_target_test_suite, _resolve_target_test_suite）：引入 _default_target_test_suite 和 _resolve_target_test_suite，根据 ROCm 设备名自动选择测试套件（MI250/MI300/MI325/MI355/L4），并支持 VLLM_TARGET_TEST_SUITE 新环境变量，保留旧 TARGET_TEST_SUITE 兼容。参数化改用 ALL_DISTRIBUTED_TEST_SUITES 和 GENERIC_DISTRIBUTED_TEST_SUITES 元组，运行时只跑与被选中套件对应的用例。
- tests/conftest.py（模块 测试配置；类别 test；类型 test-coverage）：修改 HfRunner.__enter__ 在 ROCm 下记录起始 GPU 内存使用，写入 self.threshold_ratios；修改 __exit__ 将 threshold_ratio 传递给 wait_for_rocm_memory_to_settle，并在退出时清理属性。
- .buildkite/test-amd.yaml（模块 CI 配置；类别 config；类型 configuration）：更新 CI 命令中的 TARGET_TEST_SUITE 值，使 MI250 队列使用 MI250，MI300 队列使用 MI300，MI325 队列使用 MI300（MI325 测试改用 MI300 套件），确保各队列运行对应硬件的测试参数。

关键符号：get_physical_device_indices, record_gpu_memory_usage_stats, wait_for_gpu_memory_to_clear, _default_target_test_suite, _resolve_target_test_suite

关键源码片段

tests/utils.py

重构 wait_for_gpu_memory_to_clear 支持 per-device 阈值，新增 record_gpu_memory_usage_stats 辅助函数，并为 get_physical_device_indices 添加类型注解。

```
@_nvml()
def record_gpu_memory_usage_stats(
    *,
    devices: list[int],
) -> dict[int, tuple[float, float]]:
    """
    记录指定设备的 GPU 内存用量（已用/总计），用于后续阈值等计算。
    CUDA 和 ROCm 分别使用 nvml / amdsmi API 以减小 PyTorch context 干扰。
    """
    output: dict[int, tuple[float, float]] = {}
    for device in devices:
        if current_platform.is_rocm():
            dev_handle = amdsmi_get_processor_handles()[device]
            mem_info = amdsmi_get_gpu_vram_usage(dev_handle)
            gb_used = mem_info["vram_used"] / 2**10 # amdsmi 单位 MiB → GiB
```

```
            gb_total = mem_info["vram_total"] / 2**10
        else:
            dev_handle = nvmlDeviceGetHandleByIndex(device)
            mem_info = nvmlDeviceGetMemoryInfo(dev_handle)
            gb_used = mem_info.used / 2**30
            gb_total = mem_info.total / 2**30
        output[device] = (gb_used, gb_total)
    return output


def wait_for_gpu_memory_to_clear(
    *,
    devices: list[int],
    threshold_bytes: int | dict[int, int] | None = None,
    threshold_ratio: float | dict[int, float] | None = None,
    timeout_s: float = 120,
) -> None:
    """
    等待指定 GPU 设备内存用量低于阈值（支持 per-device 独立阈值）。
    threshold_bytes / threshold_ratio 可以是单一数值（统一作用于所有设备）
    或 dict 映射（设备 -> 阈值）。
    """
    assert threshold_bytes is not None or threshold_ratio is not None
    devices = get_physical_device_indices(devices)

    # 将单一数值扩展为 per-device dict
    if isinstance(threshold_bytes, int):
        threshold_bytes = {device: threshold_bytes for device in devices}
    elif isinstance(threshold_bytes, dict):
        assert threshold_bytes.keys() == set(devices)

    if isinstance(threshold_ratio, float):
        threshold_ratio = {device: threshold_ratio for device in devices}
    elif isinstance(threshold_ratio, dict):
        assert threshold_ratio.keys() == set(devices)

    # ROCm 粗心：即使 ratio 很小，也保证至少有 4 GiB 的硬下限
    if current_platform.is_rocm() and threshold_ratio is not None:
        MIN_THRESHOLD_B = 4 * 1024**3
        if threshold_bytes is None:
            threshold_bytes = {}
        for device, ratio in threshold_ratio.items():
            threshold_bytes[device] = max(
                threshold_bytes.get(device, 0),
                MIN_THRESHOLD_B if ratio < 0.05 else 0,
            )
    # 后续循环使用 record_gpu_memory_usage_stats 检查每设备状态
```

tests/basic_correctness/test_basic_correctness.py

引入 _default_target_test_suite 和 _resolve_target_test_suite，根据 ROCm 设备名自动选择测试套件（MI250/MI300/MI325/MI355/L4），并支持 VLLM_TARGET_TEST_SUITE 新环境变量，保留旧 TARGET_TEST_SUITE 兼容。参数化改用 ALL_DISTRIBUTED_TEST_SUITES 和 GENERIC_DISTRIBUTED_TEST_SUITES 元组，运行时只跑与被选中套件对应的用例。

```
TARGET_TEST_SUITE_ENV = "VLLM_TARGET_TEST_SUITE"
LEGACY_TARGET_TEST_SUITE_ENV = "TARGET_TEST_SUITE"

GENERIC_DISTRIBUTED_TEST_SUITES = ("L4", "MI250", "MI300", "MI325", "MI355")
ALL_DISTRIBUTED_TEST_SUITES = (*GENERIC_DISTRIBUTED_TEST_SUITES, "A100")


def _default_target_test_suite() -> str:
    """依据当前 ROCm 硬件返回默认测试套件名（非 ROCm 返回 L4）。"""
    if not current_platform.is_rocm():
        return "L4"

    try:
        device_name = current_platform.get_device_name().upper()
    except Exception:
        device_name = ""

    if "MI355" in device_name:
        return "MI355"
    if "MI300" in device_name:
        return "MI300"
    if "MI325" in device_name:
        return "MI325"
    if "MI250" in device_name:
        return "MI250"

    # 若通过设备名无法识别，尝试直接查询 ROCm 平台函数
    try:
        from vllm.platforms import rocm as rocm_platform
        if rocm_platform.on_gfx950():
            return "MI355"
        if rocm_platform.on_gfx942():
            return "MI300"
    except Exception:
        pass

    # 兜底：假设为 MI250（最老架构）
    return "MI250"


def _resolve_target_test_suite() -> str:
    """
    优先级：VLLM_TARGET_TEST_SUITE > TARGET_TEST_SUITE > 自动检测。
```

```
    环境变量值会转为大写并去除首尾空白。
    """
    for env_name in (TARGET_TEST_SUITE_ENV, LEGACY_TARGET_TEST_SUITE_ENV):
        value = os.environ.get(env_name, "").strip().upper()
        if value:
            return value
    return _default_target_test_suite()


# 模块加载时解析一次，在 pytest 收集阶段确定套件
TARGET_TEST_SUITE = _resolve_target_test_suite()
```

tests/conftest.py

修改 HfRunner.__enter__ 在 ROCm 下记录起始 GPU 内存使用，写入 self.threshold_ratios；修改 __exit__ 将 threshold_ratio 传递给 wait_for_rocm_memory_to_settle，并在退出时清理属性。

```
class HfRunner:
    # ...
    def __enter__(self):
        # ROCm 下记录起始内存占用，防止前一个测试残留干扰
        if current_platform.is_rocm():
            from tests.utils import (
                get_physical_device_indices,
                record_gpu_memory_usage_stats,
            )
            if (device_count := current_platform.device_count()) > 0:
                devices = get_physical_device_indices(
                    devices=list(range(device_count))
                )
                mem_usage_stats = record_gpu_memory_usage_stats(devices=devices)
                # 阈值 = 5% 缓冲 + 当前已用比例，确保关闭时能释放回该水平
                self.threshold_ratios = {
                    device: 0.05 + mem_used / mem_tot
                    for device, (mem_used, mem_tot) in mem_usage_stats.items()
                }
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        from tests.utils import wait_for_rocm_memory_to_settle

        del self.model
        cleanup_dist_env_and_memory()
        # 使用记录的阈值等待内存稳定
        wait_for_rocm_memory_to_settle(
            threshold_ratio=getattr(self, "threshold_ratios", None)
        )
        if hasattr(self, "threshold_ratios"):
            del self.threshold_ratios
```

评论区精华

本 PR 无 review 评论讨论；但 Issue 评论中 @mawong-amd 提供了关于 per-device 内存阈值和 HfRunner 起始记录的补充实现，最终被合并。

- 暂无高价值评论线程

风险与影响

- 风险：
  - 自动检测失败：如果 current_platform.get_device_name() 返回未知名称或异常，_default_target_test_suite 会回退到 MI250，可能导致测试在非预期硬件上运行。
  - 环境变量优先级：新环境变量 VLLM_TARGET_TEST_SUITE 优先级高于旧 TARGET_TEST_SUITE，如果已有 CI 脚本依赖旧变量覆盖，可能行为改变。
  - 多设备阈值复杂度：wait_for_gpu_memory_to_clear 现在支持 dict 阈值，调用方需确保键集合与设备列表一致，否则会触发断言。
  - ROCm 驱动延迟：HfRunner 中记录的起始内存比例可能在并发测试中不准确，但现有逻辑已包含 0.05 的缓冲。
  - 影响：直接影响 ROCm CI 测试的有效性和稳定性。MI250、MI300、MI325、MI355 队列现在会按实际硬件筛选测试参数，避免之前全部跑 L4/A100 用例导致的部分用例未真正验证。非 ROCm 平台（CUDA）无行为变化。测试工具函数 wait_for_gpu_memory_to_clear 的接口扩展后，其他测试也可利用 per-device 阈值。
  - 风险标记：自动检测设备回退，环境变量覆盖顺序，多设备阈值复杂度，ROCm 驱动延迟

关联脉络

- PR #39238 (not provided in context, but referenced as base PR): 本 PR 从 #39238 拆分而来，专门提取 basic-correctness 套件扩展部分。