PR #7205 完整报告

verl-project/verl

[ckpt] feat: add hccl ckpt engine split_weight_chunks

合并时间：2026-07-31 10:44

原文链接：http://prhub.com.cn/verl-project/verl/pull/7205

执行摘要

- 一句话：为 HCCL 引擎添加权重分块传输支持
- 推荐动作：该 PR 值得精读，特别是 split_weight_chunks 的使用和异步化改造，展示了如何安全地处理大权重传输。关注点： 1) torch.npu.synchronize() 的放置位置是否正确； 2) 线程池资源是否会成为瓶颈； 3) 测试中 128MB bucket 的耗时影响。建议在 NPU 环境中进行充分的端到端验证。

功能与动机

HCCL 检查点引擎原先假设单个权重尺寸不超过 bucket_size，权重较大时会断言失败。本 PR 通过引入分块传输，允许超过 bucket 大小的权重被拆分为多个块进行异步传输，从而支持更大的模型权重（如 Qwen3-8B 的 embed_tokens 约 1.2GB）。同时，修复了 BroadcastOperation 在异步环境中可能阻塞事件循环的问题，并统一了设备回退逻辑以支持非 CUDA 平台（如 NPU）。

实现拆解

1. 导入与基础工具：在 verl/checkpoint_engine/hccl_checkpoint_engine.py 中引入 asyncio，并从 verl/checkpoint_engine.base 导入 merge_weight_chunks 和 split_weight_chunks，为分块传输做准备。
2. 异步化 BroadcastOperation：将 BroadcastOperation._run() 的执行改为通过 loop.run_in_executor 放入线程池，避免阻塞事件循环；wait_for_complete() 改为 await self._task，并补充文档说明 HCCL 内核入队与完成的区别。
3. 分块发送逻辑：在 send_weights 中，将原先对 (name, weight) 的同步迭代改为 async for tensor_meta, chunk in split_weight_chunks(weights, self.bucket_size)，并调整 bucket 填充逻辑，用 chunk_size 和 chunk 替换 nbytes 和 weight.view(-1).view(torch.uint8)，同时保留对 tensor_meta.name 唯一性的断言。最后，在最后一次 broadcast 后增加 torch.npu.synchronize()，确保缓冲区在 HCCL 内核完成后才被释放。
4. vLLM 设备回退统一：在 verl/workers/rollout/vllm_rollout/utils.py 中，将从 vllm.platforms 导入的 current_platform 替换为 get_device_name()，并将 update_weights_from_ipc 中的设备回退逻辑改为通用形式，以支持 NPU 等平台。

5. 测试扩展：在 tests/checkpoint_engine/test_correctness_on_npu.py 中，为 test_hccl_checkpoint_engine 增加了 bucket_size_mb 参数化（值 3072 和 128），以覆盖分块传输路径；同时，将 test_kimi_checkpoint_engine 和 test_mooncake_checkpoint_engine 的参数调小（如 2 个 trainer、6 个 rollout），并补充 @auto_await 标记。

关键文件：

- verl/checkpoint_engine/hccl_checkpoint_engine.py（模块 检查点引擎；类别 source；类型 core-logic；符号 _receive_weight_chunks, send_weights, BroadcastOperation）：核心逻辑文件，引入 split_weight_chunks 实现权重分块传输，并异步化 BroadcastOperation，增加同步点防止缓冲区提前释放。
- verl/workers/rollout/vllm_rollout/utils.py（模块 Rollout；类别 source；类型 dependency-wiring；符号 update_weights_from_ipc）：统一设备回退逻辑，支持 NPU 等非 CUDA 平台，与 HCCL 引擎的 NPU 支持相辅相成。
- tests/checkpoint_engine/test_correctness_on_npu.py（模块 测试；类别 test；类型 test-coverage；符号 test_hccl_checkpoint_engine）：扩展 NPU 正确性测试，覆盖分块传输路径，并调整其他测试参数以适配新逻辑。

关键符号：send_weights, BroadcastOperation._run, BroadcastOperation.wait_for_complete, update_weights_from_ipc

关键源码片段

verl/checkpoint_engine/hccl_checkpoint_engine.py

核心逻辑文件，引入 split_weight_chunks 实现权重分块传输，并异步化 BroadcastOperation，增加同步点防止缓冲区提前释放。

```
# verl/checkpoint_engine/hccl_checkpoint_engine.py

class BroadcastOperation:
    """异步广播操作，在独立线程中执行 HCCL 广播，避免阻塞事件循环。"""

    def __init__(
        self,
        rank: int,
        process_group: StatelessProcessGroup | str,
        bucket: torch.Tensor,
        metadata: dict[str, TensorMeta],
        socket: zmq.Socket,
        topic: str,
    ) -> None:
        self.rank = rank
        self.pyhccl = process_group
        self.bucket = bucket
        self.metadata = metadata
        self.socket = socket
        self.topic = topic
```

```
        # 将阻塞的 HCCL 广播放入线程池执行，避免阻塞事件循环
        loop = asyncio.get_running_loop()
        self._task = loop.run_in_executor(None, self._run)

    def _run(self):
        # rank 0 通过 ZeroMQ 发布元数据，其他 rank 接收
        if self.rank == 0:
            self.socket.send_string(self.topic, flags=zmq.SNDMORE)
            self.socket.send_pyobj(self.metadata)
        else:
            self.socket.recv_string()
            self.metadata = self.socket.recv_pyobj()

        # 通过 HCCL 广播张量数据
        self.pyhccl.broadcast(self.bucket, src=0)

    async def wait_for_complete(self) -> dict[str, TensorMeta]:
        """等待广播任务完成。

        注意：这里的完成仅指 HCCL 内核被入队，不保证内核已执行完毕，
        因此调用方在使用共享缓冲区前需要额外同步（如 torch.npu.synchronize）。
        """
        await self._task
        return self.metadata

# send_weights 中的分块传输逻辑（节选）
async def send_weights(self, weights, rank, socket, topic):
    bucket_meta = {}
    offset = 0
    # 使用 split_weight_chunks 将超大权重拆分为多个块，逐块填充发送缓冲区
    async for tensor_meta, chunk in split_weight_chunks(weights, self.bucket_size):
        if offset + tensor_meta.chunk_size > self.bucket_size:
            torch.npu.synchronize() # 等待前一个广播完成
            # 发送当前 bucket（略）
        assert offset + tensor_meta.chunk_size <= self.bucket_size
        assert tensor_meta.name not in bucket_meta
        tensor_meta.offset = offset
        bucket_meta[tensor_meta.name] = tensor_meta
        send_buf[offset : offset + tensor_meta.chunk_size] = chunk
        offset += tensor_meta.chunk_size
    # 发送最后一个 bucket
    broadcast_op = BroadcastOperation(rank, group, send_buf, bucket_meta, socket, topic)
    await broadcast_op.wait_for_complete()
    # 由于 wait_for_complete 只保证内核入队，这里显式同步确保缓冲区不被提前释放
    torch.npu.synchronize()
```

verl/workers/rollout/vllm_rollout/utils.py

统一设备回退逻辑，支持 NPU 等非 CUDA 平台，与 HCCL 引擎的 NPU 支持相辅相成。

```
# verl/workers/rollout/vllm_rollout/utils.py
```

```
class VLLMWeightSync:
    def update_weights_from_ipc(self, peft_config: dict = None, base_sync_done=False, use_shm:
    bool = False):
        """从 IPC 更新 rollout 模型权重。"""
        from verl.workers.rollout.vllm_rollout.bucketed_weight_transfer import
        BucketedWeightReceiver

        if self.device is None:
            # vLLM 工作进程可能在某些非 CUDA 平台（如 NPU）上未设置 device，
            # 这里回退到当前加速器的本地 rank，以支持 NPU 等硬件。
            self.device = torch.device(f"{get_device_name()}:{self.local_rank}")

        # ... 其余权重加载步骤
```

tests/checkpoint_engine/test_correctness_on_npu.py

扩展 NPU 正确性测试，覆盖分块传输路径，并调整其他测试参数以适配新逻辑。

```
# tests/checkpoint_engine/test_correctness_on_npu.py

@pytest.mark.asyncio
@pytest.mark.parametrize("rebuild_group", [False])
@pytest.mark.parametrize("num_trainer, num_rollout", [(2, 6)])
# 128MB bucket 小于 Qwen3-8B 最大权重（embed_tokens 约 1.2GB），
# 因此会触发分块传输路径，覆盖权重大于 bucket 的场景。
@pytest.mark.parametrize("bucket_size_mb", [3072, 128])
@auto_await
async def test_hccl_checkpoint_engine(
    rebuild_group,
    num_trainer,
    num_rollout,
    bucket_size_mb,
    num_nodes=1,
    num_gpus_per_node=8,
    check_allclose=True,
    model_path="~/models/Qwen/Qwen3-8B-Base",
):
    # ... 初始化配置，将 bucket 大小参数化，验证分块传输功能
```

评论区精华

进行中：PR 尚无实质 review 讨论，只有机器人自动评论（CLA 检查提醒）和审核人 wuxibin89 的批准。因此，讨论要点主要来自源码注释和实现逻辑：

- BroadcastOperation.wait_for_complete 的文档说明 HCCL 内核仅入队而非完成，需要额外 torch.npu.synchronize() 保护缓冲区。
- update_weights_from_ipc 中的注释解释了非 CUDA 平台（如 NPU）可能未设置 device，因此回退到通用 get_device_name()。
- 暂无高价值评论线程

风险与影响

- 风险：
  1. 异步资源泄漏风险：run_in_executor 创建的线程是全局线程池，反复调用可能累积，但 HCCL 引擎调用频率较低，风险可控。
  2. 缓冲区释放风险：send_weights 中添加的 torch.npu.synchronize() 是必要的，但若其他路径未同步，可能仍存在缓冲区过早释放的隐患。
  3. 测试环境风险：新增的 128MB bucket 测试会显著加长传输时间，可能影响 CI 稳定性；且测试仅在 NPU 上运行，无法在常规 GPU CI 中验证。
  4. 设备回退逻辑变更：update_weights_from_ipc 中移除 assert self.device is not None，改为无条件回退，可能掩盖真实设备设置错误。 - 影响：影响范围：仅限于 hccl_checkpoint_engine（NPU 场景）和 vLLM rollout 的设备初始化逻辑。对于使用 HCCL 引擎的超大模型（如 Qwen3-8B 以上），该变更解决了权重传输的阻塞问题，提升了稳定性。对常规 GPU 场景无影响，因为 get_device_name() 也能正确返回设备名。团队方面，为后续 HCCL 引擎的进一步优化（如多流传输）奠定了基础。 - 风险标记：NPU 专用测试，CI 覆盖有限，线程池资源长期占用，缓冲区同步依赖新增 synchronize

关联脉络

- PR #7184 [rollout, vllm, hardware]fix: add IPC check before invoking ipc_collect in BucketedWeightReceiver: 都涉及 NPU 平台下的权重同步，本 PR 的 device 回退逻辑与 7184 的 IPC 检查相关，共同提升 NPU 稳定性。
- PR #7161 [fsdp] refactor: move unfuse_moe_params to FSDP backend: 同为 rollouts 权重同步相关重构，且涉及 NPU 测试文件 test_correctness_on_npu.py，与本 PR 有文件关联。