PR #2205 完整报告

THUDM/slime

perf: vectorize REINFORCE++ discounted returns

合并时间：2026-08-12 13:36

原文链接：http://prhub.com.cn/THUDM/slime/pull/2205

执行摘要

- 一句话：向量化 REINFORCE++ 折扣回报计算并复用 GAE 扫描
- 推荐动作：值得精读。该 PR 展示了如何把 O(T) 串行递归安全地替换为分块并行扫描，并通过“提取通用 helper + 参考实现对照测试”控制重构风险。重点关注 chunked_discounted_returns 的 chunk 内扫描与跨 chunk 状态传播设计、右 padding/trim 的数值正确性论证，以及 chunked_gae 复用后行为是否保持。合并评审时建议补充一个真实的 context-parallel 集成测试或 GPU 冒烟测试。

功能与动机

Issue #2229 指出：get_reinforce_plus_plus_returns 对每个样本、每个 token 用 Python 反向循环计算 return，在 batched long responses 下 host 侧循环开销过大；而 ppo_utils.py 中 GAE 已经有一份 chunked discounted scan（并行前缀扫描 + 跨 chunk 递归传播）可以复用。期望在保持数值结果、变长裁剪、dtype/device 以及 context-parallel 切片行为不变的前提下，走向量化路径。PR body 提供 CPU-only 微基准：B=1、T=1024 时 21.87x，B=8、T=4096 时 357.21x，并明确说明这不是 GPU 或端到端训练吞吐声明。

实现拆解

1. 提取通用折扣扫描函数：在 slime/utils/ppo_utils.py 新增 chunked_discounted_returns(rewards, discount, chunk_size=128)，将原 chunked_gae 内部基于 FlashLinearAttention 思路的实现（chunk 内并行前缀扫描、跨 chunk 递归状态传播）抽出，使顺序依赖从 O(T) 降至 O(T / chunk_size)，chunk 内计算可并行（O(C^2) per chunk）。
2. GAE 复用同一递归：chunked_gae 改为调用 chunked_discounted_returns 完成核心扫描，再叠加 values 得到 advantages 与 returns，对外行为不变；vanilla_gae 作为串行参考实现保留。
3. 向量化 REINFORCE++ 回报路径：get_reinforce_plus_plus_returns 先按原逻辑构造每个序列的 token_level_rewards（KL 惩罚 + 末 token 叠加奖励），收集为列表后右填充成 [B, max_len] 矩阵，一次调用 chunked_discounted_returns，再逐行裁剪回原始长度，最后保持原有 all_gather_with_cp / slice_log_prob_with_cp 的 context-parallel 行为。空输入列表直接返回 []，避免 max() 崩溃。
4. 测试与 CI 配套：新增 tests/test_discounted_returns.py（45 个用例），用 _serial_discounted_returns 作为串行参考，覆盖折扣 0 / 0.5 / 0.99 / 1、序列长度 1 /

127 / 128 / 129 / 1000（跨越 128 的 chunk 边界）、float32 / float64、右 padding 零保持、REINFORCE++ 变长序列调用方以及 GAE 扫描复用；同时把 test_discounted_returns.py 登记进 .github/workflows/pr-test.yml 和 .j2 模板的 num_gpus=0 CPU 测试列表。

关键文件：

- slime/utils/ppo_utils.py（模块 回报计算；类别 source；类型 core-logic；符号 chunked_gae, chunked_discounted_returns, get_reinforce_plus_plus_returns）：核心实现文件：新增 chunked_discounted_returns，让 chunked_gae 复用同一扫描，并将 get_reinforce_plus_plus_returns 改为右填充 batch 扫描 + trim 的向量化路径。
- tests/test_discounted_returns.py（模块 回归测试；类别 test；类型 test-coverage；符号 _serial_discounted_returns, test_chunked_discounted_returns_matches_serial, test_chunked_discounted_returns_preserves_right_padding, test_reinforce_plus_plus_returns_matches_serial_for_variable_lengths）：新增 45 个 CPU 回归测试，是证明向量化与旧串行实现数值等价的依据；覆盖 chunk 边界、折扣边缘值、右 padding 保持、REINFORCE++ 调用方与 GAE 复用。
- .github/workflows/pr-test.yml（模块 CI 配置；类别 infra；类型 infrastructure）：CI 配置登记新测试 test_discounted_returns.py 为 num_gpus=0 的 CPU 任务，确保回归测试进入标准 PR 检查流程。
- .github/workflows/pr-test.yml.j2（模块 CI 配置；类别 infra；类型 infrastructure）：CI 模板同步登记新测试项，保证生成的工作流文件与模板一致。

关键符号：chunked_discounted_returns, chunked_gae, get_reinforce_plus_plus_returns

关键源码片段

slime/utils/ppo_utils.py

核心实现文件：新增 chunked_discounted_returns，让 chunked_gae 复用同一扫描，并将 get_reinforce_plus_plus_returns 改为右填充 batch 扫描 + trim 的向量化路径。

```
def get_reinforce_plus_plus_returns(
    rewards: torch.Tensor,
    kl: list[torch.Tensor],
    loss_masks: list[torch.Tensor],
    response_lengths: list[int],
    total_lengths: list[int],
    kl_coef: float,
    gamma: float,
) -> list[torch.Tensor]:
    """
    Calculates discounted returns for REINFORCE++ (https://arxiv.org/pdf/2501.03262)

    Args:
        rewards (Tensor): A tensor of scalar rewards for each sequence.
        kl (List[Tensor]): List of per-token KL divergence tensors for sequence chunks.
        loss_masks (List[Tensor]): List of response-only loss masks for each full sequence.
```

```
    response_lengths (List[int]): The full length of each response sequence.
    total_lengths (List[int]): The full length of each sequence (prompt + response).
    kl_coef (float): Coefficient for the KL penalty.
    gamma (float): The discount factor.

Returns:
    List[torch.Tensor]: A list of return (G_t) tensors for the
                        local sequence chunks owned by the current GPU rank.
"""
from megatron.core import mpu

cp_size = mpu.get_context_parallel_world_size()

# 先逐样本构造 token 级奖励，暂存列表；不再在此处算 return
token_level_rewards = []
for i in range(len(rewards)):
    local_kl_chunk = kl[i]
    total_len, response_len = total_lengths[i], response_lengths[i]

    if cp_size > 1:
        # Step 1,2: 所有 rank 收集各自 chunk 与 token_offsets，重建成完整 response
        from slime.backends.megatron_utils.cp_utils import all_gather_with_cp

        full_kl_response = all_gather_with_cp(local_kl_chunk, total_len, response_len)
    else:
        full_kl_response = local_kl_chunk

    # Step 3: 计算完整 response 上的 token 级奖励（KL 惩罚 + 末 token 叠加奖励）
    full_mask = loss_masks[i]
    assert full_mask.sum().item() > 0, f"Sequence at index {i} is fully masked."
    masked_kl = full_kl_response * full_mask
    rewards_for_seq = -kl_coef * masked_kl
    last_idx = full_mask.nonzero(as_tuple=True)[0][-1]
    rewards_for_seq[last_idx] += rewards[i]
    token_level_rewards.append(rewards_for_seq)

# 防御：没有任何有效序列时直接返回，避免 max() 崩溃
if not token_level_rewards:
    return []

# 右填充到 batch 内最长序列，一次扫描整批，避免逐样本 Python 循环
max_len = max(rewards_for_seq.size(0) for rewards_for_seq in token_level_rewards)
padded_rewards = token_level_rewards[0].new_zeros(len(token_level_rewards), max_len)
for i, rewards_for_seq in enumerate(token_level_rewards):
    padded_rewards[i, : rewards_for_seq.size(0)] = rewards_for_seq

# G_t = r_t + gamma * G_{t+1}，由 chunked_discounted_returns 以并行扫描完成
padded_returns = chunked_discounted_returns(padded_rewards, gamma)
```

```
    final_returns_chunks = []
    for i, returns_for_seq in enumerate(padded_returns):
        # padding 区域贡献为零，裁剪回原始长度即可
        returns_for_seq = returns_for_seq[: token_level_rewards[i].size(0)]
        if cp_size > 1:
            from slime.backends.megatron_utils.cp_utils import slice_log_prob_with_cp

            total_len, response_len = total_lengths[i], response_lengths[i]
            # Step 4: 取回本 rank 对应局部 chunk 的结果
            local_returns_chunk = slice_log_prob_with_cp(returns_for_seq, total_len, response_len)
        else:
            local_returns_chunk = returns_for_seq

        final_returns_chunks.append(local_returns_chunk)

    return final_returns_chunks
```

tests/test_discounted_returns.py

新增 45 个 CPU 回归测试，是证明向量化与旧串行实现数值等价的依据；覆盖 chunk 边界、折扣边缘值、右 padding 保持、REINFORCE++ 调用方与 GAE 复用。

```
def _serial_discounted_returns(rewards: torch.Tensor, discount: float) -> torch.Tensor:
    # 串行参考实现：严格按 G_t = r_t + discount * G_{t+1} 反向递推，
    # 作为 chunked 并行扫描的数值基准。
    returns = torch.zeros_like(rewards)
    running_return = torch.zeros(rewards.size(0), device=rewards.device, dtype=rewards.dtype)
    for t in reversed(range(rewards.size(1))):
        running_return = rewards[:, t] + discount * running_return
        returns[:, t] = running_return
    return returns


@pytest.mark.parametrize("discount", [0.0, 0.5, 0.99, 1.0])
@pytest.mark.parametrize("batch_size,sequence_length", [(1, 1), (3, 127), (3, 128), (3, 129), (3, 1000)])
def test_chunked_discounted_returns_matches_serial(discount, batch_size, sequence_length):
    torch.manual_seed(0)
    rewards = torch.randn(batch_size, sequence_length)
    expected = _serial_discounted_returns(rewards, discount)
    actual = chunked_discounted_returns(rewards, discount)
    torch.testing.assert_close(actual, expected, atol=1e-4, rtol=1e-5)
    # 类型和设备必须原样保留
    assert actual.dtype == rewards.dtype
    assert actual.device == rewards.device
```

评论区精华

该 PR 没有正式 review 评论（review_comments_count=0），Issue #2229 评论区仅有一条 Codex 用量提示和一条与实现无关的闲聊。技术权衡主要体现在 PR body 与测试设计中：

- 数值等价策略：CPU 测试对照旧的反向递归实现，float32 用 atol=1e-4 / rtol=1e-5，float64 用 1e-10，说明并行扫描的浮点运算顺序变化被显式允许并验证。
- 变量长度处理：右填充位置保持为 0，且测试断言 count_nonzero(padding)==0，确保 trim 前后语义一致。
- CP 行为声明：PR body 明确“不改变 context-parallel gather/slice 行为”，但测试仅在 cp_size=1 下覆盖。
  - 暂无高价值评论线程

风险与影响

- 风险：
  1. 浮点数值差异：并行 chunk 扫描改变了累加顺序，float32 下与旧串行实现的微小误差可能累积；测试容差 1e-4 且只覆盖随机正态输入，极端奖励分布下误差未验证。
  2. 右填充计算浪费：padded_rewards 按 batch 内最长序列构造，若序列长度差异悬殊，短序列的 padding 区域也会参与扫描，产生额外计算；虽然远小于 Python 循环开销，但 GPU 上的实际影响未测量。
  3. CP 路径缺少 CI 验证：get_reinforce_plus_plus_returns 的 context-parallel 分支（slice_log_prob_with_cp）在测试中通过 monkeypatch 只走了 cp_size=1，真实多 rank CP 行为没有自动测试。
  4. 空输入新分支：新增 if not token_level_rewards: return []，属于防御性改动，但上层调用方是否依赖旧行为（如空列表时报错）未在测试中体现。
- 影响：影响面集中在 RL 训练数值链路：
  - 用户 / 算法侧：REINFORCE++ 折扣回报计算结果与之前保持一致（测试验证），但 CPU host 端计算耗时大幅下降，长响应大 batch 场景的训练吞吐有望改善；GAE 路径通过复用 chunked_discounted_returns 也间接受益于统一实现。
  - 系统侧：新增通用函数 chunked_discounted_returns 成为折扣扫描的单一实现点，后续其他算法（如 ReMax、GRPO 类）可直接复用。
  - 团队 / 工程侧：新增独立测试文件并接入 CI（num_gpus=0），回归门槛清晰；但 context-parallel 分支仍未纳入自动化验证。
  - 风险标记：并行扫描浮点顺序变化需容差验证，CP 分支仅有 cp_size=1 测试覆盖，右填充按最长序列计算可能浪费，性能收益仅在 CPU 微基准验证

关联脉络

- PR #2235 fix: whiten advantages over the DP group that includes context parallel: 同属 RL 训练数值路径的修复，均涉及 Megatron 并行组与 advantage/return 计算的一致性，可互相参考验证。
- PR #2247 fix: forward dual-clip PPO epsilon: 同为 PPO/ 策略损失数值链路的修复，与本次 on-policy 回报计算都属于训练核心路径，后续改动需保持数值行为一致。