PR #6526 完整报告

verl-project/verl

[megatron] feat: align optimizer states and DDP grad bucket with model precision

合并时间：2026-06-23 12:06

原文链接：http://prhub.com.cn/verl-project/verl/pull/6526

执行摘要

- 一句话：Megatron 优化器状态与 DDP 梯度桶对齐模型类型
- 推荐动作：该 PR 值得精读，尤其是精度感知优化器与 DDP 梯度桶的联动设计。对于关注显存优化的团队，建议评估后启用，并在各自任务上进行收敛性验证。

功能与动机

在 bf16 训练中，Megatron 分布式优化器保持 Adam 动量和梯度累积缓冲在 fp32，占用了不必要的显存（3 倍所需）。本 PR 使其可选地遵循模型 dtype，类似于 Megatron CLI 的 `--use-precision-aware-optimizer` 等选项。最终经过讨论和实验验证，确定为可配置选项，默认保持 fp32 以避免收敛风险。

实现拆解

1. 重构 init_megatron_optim_config（verl/utils/megatron/optimizer.py）：添加 bf16 参数（默认 True），将原先的 *else* 分支拆分为 *elif* bf16:（bf16 模式）和 *else*:（fp32 模式）。在 *elif* bf16: 内部，当检测到 use_precision_aware_optimizer=True 时，启用精度感知优化器，将 main_grads_dtype、*exp_avg_dtype*、*exp_avg_sq_dtype* 设置为对应 dtype（利用 PrecisionType.to_dtype 转换）；fp32 模式则禁用精度感知优化器。fp16 分支保持不变。
2. 扩展 McoreOptimizerConfig（verl/workers/config/optimizer.py）：新增 use_precision_aware_optimizer（默认 False）、main_grads_dtype（默认 "fp32"）、*exp_avg_dtype*（默认 "fp32"）、*exp_avg_sq_dtype*（默认 "fp32"）字段，并添加 __post_init__ 验证 dtype 字符串合法性。同步更新 verl/trainer/config/optim/megatron.yaml 和 _generated_ppo_megatron_trainer.yaml 默认配置。
3. 引擎集成（verl/workers/engine/megatron/transformer_impl.py）：新增 _resolve_override_ddp_config 方法，根据优化器配置判断是否需将 grad_reduce_in_fp32 设为 False 以同步 DDP 梯度桶 dtype；用户显式设置时优先。在 _build_megatron_module 中调用该方法，并在 _build_optimizer 调用 init_megatron_optim_config 时传入 bf16 参数。
4. 其他调用点更新：megatron_workers.py 中的调用已随文件删除而移除（冲突解决后仅保留 transformer_impl.py 的更新）。

5. 新增测试：tests/utils/megatron/test_optimizer.py 包含 9 个 CPU 单元测试，通过 monkeypatchOptimizerConfig 为记录器验证各模式下组装的 kwargs，覆盖 bf16 默认 fp32、bf16 启用精度感知、每字段独立控制、fp16 分支、fp32 分支、默认参数传递、override 优化器配置等场景。

关键文件：

- tests/utils/megatron/test_optimizer.py（模块 测试；类别 test；类型 test-coverage；符号 _base_optim_config, _precision_aware_optim_config, captured_args, _fake）：9 个 CPU 单元测试验证各模式下配置组装，确保逻辑正确性
- verl/utils/megatron/optimizer.py（模块 优化器；类别 source；类型 dependency-wiring）：核心函数 init_megatron_optim_config 的重构，增加 bf16 参数和精度感知分支
- verl/workers/engine/megatron/transformer_impl.py（模块 引擎；类别 source；类型 core-logic；符号 _resolve_override_ddp_config）：新增 _resolve_override_ddp_config 确保 DDP 梯度桶与精度感知优化器一致，并更新 _build_optimizer 调用

关键符号：init_megatron_optim_config, McoreOptimizerConfig.post_init, _resolve_override_ddp_config, _base_optim_config, _precision_aware_optim_config, test_bf16_branch_defaults_to_fp32_optimizer_state, test_bf16_branch_opt_in_enables_precision_aware_with_bf16_state, test_bf16_opt_in_respects_per_field_dtypes, test_fp16_branch_uses_precision_aware_but_keeps_fp32_optimizer_state

关键源码片段

tests/utils/megatron/test_optimizer.py

9 个 CPU 单元测试验证各模式下配置组装，确保逻辑正确性

```
import pytest
import torch
from omegaconf import OmegaConf
from unittest.mock import MagicMock
from verl.utils.megatron import optimizer as opt_mod
from verl.utils.megatron.optimizer import init_megatron_optim_config

def _base_optim_config(**overrides):
    # 基础优化配置字典（OmegaConf）
    cfg = {
        "optimizer": "adam",
        "lr": 1e-3,
        "min_lr": 0.0,
        "clip_grad": 1.0,
        "weight_decay": 0.01,
    }
    cfg.update(overrides)
    return OmegaConf.create(cfg)

def _precision_aware_optim_config(**overrides):
```

```
    # 启用精度感知优化器的配置
    fields = {
        "use_precision_aware_optimizer": True,
        "main_grads_dtype": "bf16",
        "exp_avg_dtype": "bf16",
        "exp_avg_sq_dtype": "bf16",
    }
    fields.update(overrides)
    return _base_optim_config(**fields)

@pytest.fixture
def captured_args(monkeypatch):
    # 替换 OptimizerConfig 为记录器，捕获实际传入的 kwargs
    captured: dict = {}
    def _fake(**kwargs):
        captured.clear()
        captured.update(kwargs)
        return MagicMock(name="OptimizerConfig", **kwargs)
    monkeypatch.setattr(opt_mod, "OptimizerConfig", _fake)
    return captured

def test_bf16_branch_defaults_to_fp32_optimizer_state(captured_args):
    # 默认 bf16 模式下优化器状态保持 fp32（精度感知未启用）
    init_megatron_optim_config(_base_optim_config(), fp16=False, bf16=True)
    assert captured_args["bf16"] is True
    assert captured_args["params_dtype"] is torch.bfloat16
    # 不应出现精度感知相关的键
    assert "use_precision_aware_optimizer" not in captured_args
    assert "main_grads_dtype" not in captured_args
    assert "exp_avg_dtype" not in captured_args
    assert "exp_avg_sq_dtype" not in captured_args

def test_bf16_branch_opt_in_enables_precision_aware_with_bf16_state(captured_args):
    # 显式启用精度感知优化器后，main_grads_dtype 等应变为 bf16
    init_megatron_optim_config(
        _precision_aware_optim_config(), fp16=False, bf16=True
    )
    assert captured_args["use_precision_aware_optimizer"] is True
    assert captured_args["main_grads_dtype"] is torch.bfloat16
    assert captured_args["exp_avg_dtype"] is torch.bfloat16
    assert captured_args["exp_avg_sq_dtype"] is torch.bfloat16
    # master_params_dtype 保留 fp32（TE FusedAdam 限制）
    assert "main_params_dtype" not in captured_args
```

verl/utils/megatron/optimizer.py

核心函数 init_megatron_optim_config 的重构，增加 bf16 参数和精度感知分支

```
import torch
from megatron.core.optimizer import OptimizerConfig
```

```
from verl.utils.logger import print_rank_0
from verl.utils.torch_dtypes import PrecisionType

def init_megatron_optim_config(
    optim_config: dict,
    use_distributed_optimizer: bool = True,
    fp16: bool = False,
    bf16: bool = True, # 默认 True 对应 bf16 模式，但精度感知优化器默认关闭
) -> OptimizerConfig:
    # 基础配置项
    optim_args = {
        "optimizer": optim_config.optimizer,
        "lr": optim_config.lr,
        "min_lr": optim_config.min_lr,
        "clip_grad": optim_config.clip_grad,
        "weight_decay": optim_config.weight_decay,
        "use_distributed_optimizer": use_distributed_optimizer,
    }
    if fp16:
        # fp16 分支：启用精度感知优化器，但 Adam 动量保持 fp32（默认）
        optim_args.update({
            "bf16": False,
            "fp16": True,
            "params_dtype": torch.float16,
            "initial_loss_scale": 32768,
            "min_loss_scale": 1,
            "use_precision_aware_optimizer": True,
            "store_param_remainders": False,
        })
    elif bf16:
        # bf16 分支：模型参数为 bf16，但优化器状态默认 fp32（向后兼容）
        optim_args.update({
            "bf16": True,
            "params_dtype": torch.bfloat16,
        })
        # 精度感知优化器为可选项：仅在配置中设置 use_precision_aware_optimizer=True 后启用
        if optim_config.get("use_precision_aware_optimizer", False):
            optim_args.update({
                "use_precision_aware_optimizer": True,
                # 从配置中读取各字段类型，默认 fp32（即不降低精度）
                "main_grads_dtype": PrecisionType.to_dtype(
                    optim_config.get("main_grads_dtype", "fp32")
                ),
                "exp_avg_dtype": PrecisionType.to_dtype(
                    optim_config.get("exp_avg_dtype", "fp32")
                ),
                "exp_avg_sq_dtype": PrecisionType.to_dtype(
                    optim_config.get("exp_avg_sq_dtype", "fp32")
                ),
```

```
            })
    else:
        # fp32 分支：禁用精度感知优化器（Megatron 要求关闭时 dtype 必须为 fp32）
        optim_args.update({
            "bf16": False,
            "fp16": False,
            "params_dtype": torch.float32,
        })
    # 允许用户通过 override_optimizer_config 完全覆盖最终的任何配置项
    override_config = optim_config.get("override_optimizer_config", {})
    if override_config:
        for k, v in override_config.items():
            optim_args[k] = v
    print_rank_0(f"optimizer config after override: {optim_args}")
    config = OptimizerConfig(**optim_args)
    return config
```

verl/workers/engine/megatron/transformer_impl.py

新增 _resolve_override_ddp_config 确保 DDP 梯度桶与精度感知优化器一致，并更新 _build_optimizer 调用

```
  def _resolve_override_ddp_config(self):
    # 保持 DDP 梯度桶 dtype 与优化器的梯度缓冲 dtype 一致
    # 当精度感知优化器启用且 main_grads_dtype 低于 fp32 时，
    # 自动设置 grad_reduce_in_fp32=False，除非用户已显式指定
    from verl.utils.torch_dtypes import PrecisionType

    override_ddp_config = dict(self.engine_config.override_ddp_config or {})
    opt_cfg = self.optimizer_config
    if (
        opt_cfg is not None
        and getattr(opt_cfg, "use_precision_aware_optimizer", False)
        and PrecisionType.to_dtype(getattr(opt_cfg, "main_grads_dtype", "fp32")) != torch.float32
        and "grad_reduce_in_fp32" not in override_ddp_config
    ):
        override_ddp_config["grad_reduce_in_fp32"] = False
    return override_ddp_config

  def _build_megatron_module(self):
    # ... 其他代码 ...
    override_ddp_config = self._resolve_override_ddp_config() # 新增调用
    module, updated_tf_config = make_megatron_module(
        ...
        override_ddp_config=override_ddp_config, # 使用解析后的配置
        ...
    )
    # ...

  def _build_optimizer(self):
```

```
    # ...
    optim_config_megatron = init_megatron_optim_config(
        self.optimizer_config,
        use_distributed_optimizer=self.engine_config.use_distributed_optimizer,
        fp16=self.param_dtype == torch.float16,
        bf16=self.param_dtype == torch.bfloat16, # 新增：传入 bf16 标志
    )
    # ...
```

评论区精华

讨论主要围绕优化器精度是否应保持 fp32 展开：

- wuxibin89最初评论“optimizer should be in fp32”，认为优化器应保持 fp32。
- kolehma8解释可以暴露为配置，并指出 FSDP 中已有此做法。
- wuxibin89澄清 FSDP 中他们强制模型 fp32，优化器也因此 fp32，并要求实验验证低精度对收敛的影响。
- kolehma8在 Issue #6576 中发布了对比实验结果，显示在此示例中数值精度不影响奖励，建议将低精度作为可选配置而非默认。
- wuxibin89要求解决冲突（megatron_workers.py 已移除），最终批准了 PR。
- 优化器精度是否应保持 fp32 (design): 经过实验验证，决定将低精度作为可选配置，默认保留 fp32。
- 收敛性验证要求 (correctness): 实验支撑了选项的可行性，最终确定为配置而非默认。
- 冲突解决（megatron_workers.py 移除） (other): kolehma8 rebase 解决冲突，仅保留 transformer_impl.py 中的改动。

风险与影响

- 风险：
  1. 数值收敛风险：启用 bf16 优化器状态可能导致某些场景下训练不收敛，虽然实验显示无影响，但默认关闭降低了风险，用户需自行验证。
  2. DDP 梯度桶一致性风险：_resolve_override_ddp_config 确保了一致性，但依赖于 main_grads_dtype 配置。默认 main_grads_dtype 为 fp32 时不会修改 grad_reduce_in_fp32，安全。
  3. TransformerEngine 限制：TE FusedAdam 不支持 bf16 master weights，因此 main_params_dtype 保留 fp32，无法进一步省显存。
  4. 向后兼容性：默认行为未变，不影响现有用户。
  5. 测试覆盖：CPU 单元测试覆盖配置逻辑，但缺乏端到端集成测试。
- 影响：
  - 用户影响：使用 Megatron 后端的用户可通过配置降低优化器状态显存（约 1/3），默认不启用无影响。
  - 系统影响：新增配置项和内部逻辑，不改变现有训练流程。
  - 团队影响：增加维护复杂性但代码结构化清晰，测试覆盖主要分支。

- 风险标记：默认关闭，需显式启用，依赖 TE FusedAdam，需要额外验证收敛性，仅 Megatron 后端，缺乏端到端 CI 测试

关联脉络

- 暂无明显关联 PR