PR #7120 完整报告

verl-project/verl

[megatron] feat: add Muon optimizer support (expose Megatron-Core TensorParallelMuon)

合并时间：2026-07-30 12:06

原文链接：http://prhub.com.cn/verl-project/verl/pull/7120

执行摘要

- 一句话：为 Megatron 后端添加 Muon 优化器支持
- 推荐动作：此 PR 值得精读，尤其是 adamw_rms_match_scale_factor 的推导和实验对比方法。代码设计上对兼容性的处理（字段白名单转发、fail-closed）和 DDP 封装的重构（wrap_model_chunks_with_layerwise_aware_ddp）均具有参考价值。建议使用者优先配置 muon_match_adamw_update_rms 而非手动设置常量。

功能与动机

verl 的本地 Megatron 后端此前仅支持 AdamW 类优化器；而 Megatron-Core 已通过 emerging_optimizers 包提供 TensorParallelMuon，但 verl 未暴露此能力。本 PR 旨在为 verl 用户在 Megatron 引擎上提供 Muon 优化器选择，以利用其在内存和速度上的优势。

实现拆解

1. 配置层扩展：在 verl/workers/config/optimizer.py 的 McoreOptimizerConfig 中添加 optimizer='muon' 及所有 muon_* 超参字段（如 muon_momentum、muon_num_ns_steps 等），同时在训练配置 YAML 文件中暴露默认值，便于用户选用。
2. 优化器构建连线：在 verl/utils/megatron/optimizer.py 中实现 _add_muon_args 函数，将 verl 配置中的 Muon 参数转发到 Megatron-Core 的 OptimizerConfig，仅转发已声明的字段以保持向后兼容；新增 adamw_rms_match_scale_factor 根据 beta1 计算缩放因子，使 Muon 的更新幅度与 AdamW 对齐。
3. LayerWise-DDP 感知封装：在 verl/utils/megatron_utils.py 中新增 wrap_model_chunks_with_layerwise_aware_ddp 函数，为 Muon 的层级分布式优化器（LayerWiseDistributedOptimizer）进行参数标记、布局计算，替代原简单的 DDP 封装；同时添加 _assert_muon_layer_wise_ddp_supported 前置检查，在不支持的 Megatron 版本上直接报错。
4. 测试与文档：新增 5 个测试文件，覆盖 CPU 存根转发、真实 Megatron 配置验证、GPU 单步冒烟以及配置单元测试；编写 examples/muon/README.md 说明用法和推荐开关。所有修改均 fail-closed，避免静默退化。

关键文件：

- verl/utils/megatron/optimizer.py（模块 优化器连线；类别 source；类型 dependency-wiring；符号 is_muon_layer_wise_config, _add_muon_args,

adamw_rms_match_scale_factor）：核心逻辑：实现 Muon 优化器参数转发、LayerWise 配置检测和 AdamW 更新幅匹配缩放因子。

- verl/utils/megatron_utils.py（模块 DDP 封装；类别 source；类型 dependency-wiring；符号 _assert_muon_layer_wise_ddp_supported, wrap_model_chunks_with_layerwise_aware_ddp）：提供 LayerWise-DDP 感知的模型封装，是 Muon 层级优化器正确工作的前提。
- tests/utils/megatron/test_muon_optim_wiring_on_cpu.py（模块 布线测试；类别 test；类型 test-coverage；符号 _install_stub_megatron, muon_config, test_muon_fields_forwarded, test_only_supported_muon_fields_forwarded）：CPU 存根测试，验证 Muon 参数转发和 fail-closed 逻辑，不依赖真实 Megatron。
- tests/utils/megatron/test_muon_layerwise_bridge_ddp_on_cpu.py（模块 层 DDP 测试；类别 test；类型 test-coverage；符号 _DummyConfig, _DummyModel, _FakeDDPConfig, _minimal_hf_config）：验证 Megatron-Bridge 路径下 LayerWise DDP 封装是否正确调用。
- tests/utils/megatron/muon_optim_gpu_smoke.py（模块 GPU 冒烟；类别 test；类型 test-coverage；符号 main, Net, init, forward）：GPU 冒烟测试：在真实单卡上构建 Muon 优化器并执行一步更新，确认未退化为 Adam。
- tests/utils/megatron/test_muon_optim_realcfg_on_cpu.py（模块 实配置测试；类别 test；类型 test-coverage；符号 _real_megatron_optimizer_config, test_verl_wiring_builds_real_megatron_muon_config, test_verl_adam_path_leaves_real_megatron_config_standard）：使用真实 Megatron-Core OptimizerConfig 验证 verl 配置能构造出正确的 muon 配置对象。
- tests/workers/config/test_optim_config_on_cpu.py（模块 配置测试；类别 test；类型 test-coverage；符号 TestMcoreOptimizerConfigMuonCPU, test_default_optimizer_is_adam, test_muon_defaults_track_megatron, test_muon_overrides_are_carried）：验证 McoreOptimizerConfig 中 Muon 相关默认值和覆盖逻辑。
- verl/workers/config/optimizer.py（模块 配置定义；类别 source；类型 core-logic）：配置定义入口：添加 optimizer='muon' 选择及所有 muon_* 字段。
- verl/trainer/config/optim/megatron.yaml（模块 训练配置；类别 config；类型 configuration）：训练配置示例，展示 Muon 优化器及相关参数的 YAML 写法。

关键符号：is_muon_layer_wise_config, _add_muon_args, adamw_rms_match_scale_factor, _assert_muon_layer_wise_ddp_supported, wrap_model_chunks_with_layerwise_aware_ddp, init_megatron_optim_config

关键源码片段

verl/utils/megatron/optimizer.py

核心逻辑：实现 Muon 优化器参数转发、LayerWise 配置检测和 AdamW 更新幅匹配缩放因子。

```
# Copyright ... ( 保留原有头部 )

import dataclasses
```

```
import math

# 仅 'muon' 视为 Muon 算法（新引入的常数）
_MUON_ALGORITHMS = ("muon",)


def _add_muon_args(optim_args: dict, optim_config: dict) -> None:
    """将 verl 配置中的 Muon 参数转发到 Megatron OptimizerConfig。

    只有安装的 Megatron 声明的字段才会被转发，未声明的静默跳过。
    如果请求 Muon 但安装的 Megatron 没有任何 Muon 字段，则会直接报错。
    """
    # 获取当前 OptimizerConfig 所有声明字段名
    supported_fields = {f.name for f in dataclasses.fields(OptimizerConfig)}
    forwarded = []
    for field in _MUON_PASSTHROUGH_FIELDS:
        if field not in supported_fields:
            continue
        value = optim_config.get(field, None)
        if value is None:
            continue
        optim_args[field] = value
        forwarded.append(field)

    # 如果没有任何 Muon 字段被支持，则抛出 ValueError 而非静默退化为 Adam
    muon_related = supported_fields & set(_MUON_PASSTHROUGH_FIELDS)
    if not muon_related:
        raise ValueError(
            f"optimizer={optim_args['optimizer']!r} requests Muon, but the installed "
            "megatron.core.optimizer.OptimizerConfig exposes no Muon fields. Muon requires a "
            "Megatron-Core build with emerging_optimizers support; refusing to fall back to Adam.
            "
        )
    print_rank_0(f"Muon optimizer selected; forwarded fields: {forwarded}")


def adamw_rms_match_scale_factor(beta1: float) -> float:
    """根据 AdamW 的 beta1 计算 Muon 缩放因子，使得更新幅度 RMS 与 AdamW 一致。

    upstream 文档指出：当 muon_scale_mode='spectral' 时，额外缩放因子
    sqrt((1 - B1) / (1 + B1)) 可保证学习率迁移（ beta1=0.9 时约 0.229 ）。
    """
    return math.sqrt((1.0 - beta1) / (1.0 + beta1))
```

verl/utils/megatron_utils.py

提供 LayerWise-DDP 感知的模型封装，是 Muon 层级优化器正确工作的前提。

```
# 前置检查：确保安装的 Megatron-Core 支持 LayerWise 分布式优化器
def _assert_muon_layer_wise_ddp_supported() -> None:
```

```
    """Fail closed when Megatron-Core cannot build LayerWise DDP layouts."""
    try:
        from megatron.core.optimizer.layer_wise_optimizer import (
            LayerWiseDistributedOptimizer,
            tag_params_for_buffer_routing,
        )
    except ImportError as exc:
        raise ValueError(
            "Muon layer-wise distributed optimizer requires Megatron-Core "
            "layer_wise_optimizer support. Upgrade megatron-core or disable "
            "use_layer_wise_distributed_optimizer."
        ) from exc
    try:
        DistributedDataParallelConfig(use_layer_wise_param_layout=True)
    except TypeError as exc:
        raise ValueError(
            "Muon layer-wise distributed optimizer requires DistributedDataParallelConfig."
            "use_layer_wise_param_layout. Upgrade megatron-core or disable "
            "use_layer_wise_distributed_optimizer."
        ) from exc

def wrap_model_chunks_with_layerwise_aware_ddp(
    model_chunks,
    tfconfig,
    *,
    use_distributed_optimizer: bool = True,
    use_layer_wise_distributed_optimizer: bool = False,
    override_ddp_config: dict | None = None,
):
    """封装模型块为 DDP，若启用 layer-wise 则提前标记参数并计算完整布局。

    模仿 Megatron 自带的 wrap_model_chunks_with_ddp，确保 Muon 的
    tag_params_for_buffer_routing 和 compute_full_param_layout 在
    DDP 构建前执行，避免冗余缓冲区。
    """
    from megatron.core.distributed import DistributedDataParallel as DDP
    from megatron.core.optimizer.layer_wise_optimizer import (
        LayerWiseDistributedOptimizer,
        tag_params_for_buffer_routing,
    )
    from megatron.core.process_groups_config import ProcessGroupCollection
    from megatron.core.utils import get_pg_size

    ddp_config_dict = {
        "use_distributed_optimizer": use_distributed_optimizer,
        "grad_reduce_in_fp32": True,
        "overlap_grad_reduce": False,
    }
    if override_ddp_config is not None:
```

```
        ddp_config_dict.update(override_ddp_config)
    ddp_config = DistributedDataParallelConfig(**ddp_config_dict)

    per_chunk_layouts = [None] * len(model_chunks)
    if use_layer_wise_distributed_optimizer:
        tag_params_for_buffer_routing(model_chunks)
        # LayerWise 路径要求使用 DistOpt 样式的布局（处理标量缓冲区）
        ddp_config.use_distributed_optimizer = True
        if ddp_config_dict.get("use_layer_wise_param_layout") is None:
            ddp_config.use_layer_wise_param_layout = True
        layout_pgs = ProcessGroupCollection.use_mpu_process_groups()
        assert layout_pgs.dp_cp is not None, "dp_cp process group required for LayerWise param
        layout"
        dp_size = get_pg_size(layout_pgs.dp_cp)
        expt_dp_size = get_pg_size(getattr(layout_pgs, "expt_dp", None))
        for i, chunk in enumerate(model_chunks):
            all_params = [p for p in chunk.parameters() if p.requires_grad]
            per_chunk_layouts[i] = LayerWiseDistributedOptimizer.compute_full_param_layout(
                all_params,
                ddp_config.bucket_size,
                dp_size,
                ddp_config,
                expert_data_parallel_world_size=expt_dp_size,
            )

    # 根据布局构造 DDP 模型（略：后续循环使用 per_chunk_layouts 创建 DDP 对象）
```

评论区精华

PR 作者在 body 中详细讨论了 muon_extra_scale_factor 的重要性：Muon 默认缩放因子 1.0 会导致有效步长约为 AdamW 的 4.4 倍，可能引发不稳定。为此在第 5 个提交中新增 muon_match_adamw_update_rms 开关，基于 beta1 自动推导缩放因子。第 6 个提交进一步强调应配置该开关而非手工输入常数。实验设计方面，作者通过同配置 AdamW 对比组（n=2）量化了运行间方差，确认 Muon 的内存减少（18.3%）和速度提升（约 2x）远超方差范围，而 reward 差异在方差内，说明 Muon 与 AdamW 在效果上持平。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 兼容性风险：Muon 需要 Megatron-Core 携带 emerging_optimizers；通过 _add_muon_args 中的 fail-closed（检测 OptimizerConfig 字段）在缺失时直接报错，避免静默退化。
  2. LayerWise-DDP 路径复杂度：wrap_model_chunks_with_layerwise_aware_ddp 引入了依赖（tag_params_for_buffer_routing、compute_full_param_layout），在非标准模型或未来 Megatron 版本中可能失败；现有 CPU 测试仅验证逻辑，未覆盖多卡交互。

3. Scale factor 风险：若用户直接设置 muon_extra_scale_factor 而非使用 muon_match_adamw_update_rms，可能导致训练不稳定；此风险已通过文档和推荐开关缓解。
4. 测试覆盖缺口：GPU 冒烟仅单卡，未验证多 TP/PP 场景下 Muon 优化器的实际构建和工作正确性。 - 影响：用户影响：使用 Megatron 引擎的用户现在可以在配置中选择 optimizer=muon，并结合 muon_match_adamw_update_rms 等开关。系统影响：优化器峰值内存减少约 18%（30B 模型），更新步骤加速约 2x；端到端吞吐受 token 方差影响不显著。团队影响：新增约 400 行核心源码和 500 行测试，需维护 Muon 特定连线路径；fail-closed 设计降低了版本兼容负担。影响的用户群体是 Megatron 后端用户（相对较小但有特殊需求），整体影响程度为中等。 - 风险标记：兼容性要求新 Megatron-Core 版本，LayerWise-DDP 路径复杂度，Scale factor 设置影响训练稳定性，GPU 测试仅单卡未覆盖多卡

关联脉络

- PR #7101 [docker] feat: upgrade vllm and megatron version, add packages to support DeepSeek-V4: Muon 优化器依赖 Megatron-Core 的 emerging_optimizers 包，该 PR 升级了 Megatron 版本并包含相关依赖。