PR #1572 完整报告

radixark/miles

[optim]--rematerialize-param-from-master-weight: save the bf16 weight backup in colocate

合并时间：2026-08-07 08:45

原文链接：http://prhub.com.cn/radixark/miles/pull/1572

执行摘要

- 一句话：colocate 下从主权重重建 bf16 备份，省 129 GB 内存 / 节点
- 推荐动作：值得精读。设计上以“master cast + param all-gather 与 step end 逐位相同”作为理论依据，用 init-time 覆盖断言和 opt-in SHA256 校验兜底，是“用计算换内存”的安全范式。建议重点关注 _TensorBackuperMainCast 的 lifecycle 设计、_assert_rematerialize_coverage 的覆盖判据（DDP buffer membership 在 DP>1 下的正确性论证），以及 _validate_rematerialize_param_from_master_weight 中每条 assert 背后的配置组合心智模型。后续跟进 #1588 解除 critic-only fence 与异常安全 / 分布式失败标志的补强。

功能与动机

PR body 明确指出：colocate 下 weights_backuper 会为每个 rank 保存 2 bytes/param 的 pinned CPU bf16 副本，在 GLM-5.2 744B 上单 rank 达 26-37.5 GB、单节点 104-150 GB，挤占了实测 858/898 GiB 的 rollout 峰值预算。而 update_weights 本就读取 live GPU 权重，wake 时又能通过 _copy_main_params_to_model_params + param all-gather 逐位重建 step 末的 bf16 权重，因此这份完整 CPU 备份是冗余的。

实现拆解

实现按 5 步展开：

1. 新增 CLI 参数与校验：在 miles/utils/arguments.py 中新增 --rematerialize-param-from-master-weight 与 --check-rematerialize-param-from-master-weight，并新增 _validate_rematerialize_param_from_master_weight。校验先 guard 后端（避免 FSDP Namespace 上的 AttributeError），随后拒绝 LoRA、--debug-disable-optimizer、--indep-dp、非 colocate、offload 到 disk、非分布式优化器、--keep-old-actor、无 optimizer-cpu-offload 的 precision-aware 优化器、--overlap-param-gather、关闭 advantages/returns、critic-only warmup 等组合；--debug-train-only 会静默关闭该特性；CI 模式下自动开启 SHA256 校验。
2. 备份器抽象扩展：在 miles/utils/tensor_backper.py 中新增 frozen dataclass MainCastContext 与 _TensorBackuperMainCast 变体，TensorBackuper.create 增加 main_cast_ctx 参数决定工厂分发。新变体对 actor 标签只备份 extras（expert_bias buffer 与 fp32 参数），restore 时调用 ctx 内的 cast 闭包 + start_param_sync 重建权

重；ref/teacher 等非 actor 标签没有 master 可重建，委托内部 _TensorBackuperNormal 保留完整 pinned 副本。

3. 重建逻辑封装：新增 miles/backends/megatron_utils/rematerialize_utils.py，提供 build_main_cast_context 组装 MainCastContext；_build_cast_main_to_params_fn 区分普通分布式优化器（逐一调用 chained_optimizers 的 _copy_main_params_to_model_params）与 precision-aware + cpu-offload 的 HybridDeviceOptimizer（通过 _replay_hybrid_device_copy_back 重放 HDO 的两个 step post-hook）；_assert_rematerialize_coverage 断言每个 named parameter 要么在 DDP/expert-parallel buffer 中、要么在 extras 备份中，否则启动即失败。
4. Actor 生命周期接入：在 miles/backends/megatron_utils/actor.py 中，sleep 时若启用该 flag 且角色为 actor，只 pause grad_buffer 而保留 param buffer；update_weights 结束后再 pause param_buffer，保证 weight-sync 期间 bf16 参数仍可被读取。
5. 测试配套：新增 tests/fast/utils/test_tensor_backper.py 与 tests/fast/backends/megatron_utils/test_rematerialize_utils.py，覆盖 bit 级往返、owned-shard 语义、chained optimizer 逐个 cast、HDO 双 map 重放与优先级、SHA256 校验的前两 cycle 与 tensor set 变化；tests/fast/utils/test_arguments.py 增加 TestValidateRematerializeParamFromMasterWeight 的接受 / 拒绝参数化矩阵。e2e 侧在 11 个 colocate 测试（Qwen3-30B-A3B、Qwen3.5-35B、mimo-7B、gsm8k）中启用该 flag，验证 MCore 与 HDO 两条路径。

关键文件：

- miles/backends/megatron_utils/rematerialize_utils.py（模块 重建逻辑；类别 source；类型 core-logic；符号 build_main_cast_context, _build_cast_main_to_params_fn, _replay_hybrid_device_copy_back, _assert_rematerialize_coverage）：新增模块，承载从 fp32 master 重建低精度权重的完整逻辑：MainCastContext 组装、MCore/HDO 两条 cast 路径、覆盖断言。
- miles/utils/tensor_backper.py（模块 备份器；类别 source；类型 dependency-wiring；符号 MainCastContext, create, _TensorBackuperMainCast）：备份器核心抽象扩展，新增 MainCastContext 与 _TensorBackuperMainCast，是内存节省的直接载体。
- miles/utils/arguments.py（模块 参数校验；类别 source；类型 core-logic；符号 _validate_rematerialize_param_from_master_weight）：新增 CLI 开关与严谨的兼容性断言，防止不兼容配置在启动后期崩溃。
- miles/backends/megatron_utils/actor.py（模块 执行器；类别 source；类型 dependency-wiring；符号 sleep, update_weights）：sleep/update_weights 生命周期接入：保持 param buffer 常驻到 update_weights 后 pause，是内存时序的关键。
- tests/fast/utils/test_tensor_backper.py（模块 备份器测试；类别 test；类型 test-coverage；符号 test_round_trip_restores_bit_identical_weights, test_restore_only_covers_owned_shard_and_relies_on_param_sync, test_check_verifies_first_cycles_and_raises_on_corruption, test_non_actor_tag_keeps_full_pinned_copy）：对 _TensorBackuperMainCast 的 bit 级往返、owned-shard、chained optimizer、SHA256 校验等行为做字节级锁定。

- tests/fast/backends/megatron_utils/test_rematerialize_utils.py（模块 重建测试；类别 test；类型 test-coverage；符号 test_mcore_cast_calls_every_chained_optimizer, test_hdo_replay_covers_both_fractions, test_hdo_replay_cpu_hook_takes_precedence_on_overlap）：验证 MCore 与 HybridDeviceOptimizer 两条 cast 路径的构建与重放语义。

关键符号：build_main_cast_context, _build_cast_main_to_params_fn, _replay_hybrid_device_copy_back, _assert_rematerialize_coverage, _named_restore_extras, _TensorBackuperMainCast.backup, _TensorBackuperMainCast.restore, _TensorBackuperMainCast.get, TensorBackuper.create, _validate_rematerialize_param_from_master_weight

关键源码片段

miles/backends/megatron_utils/rematerialize_utils.py

新增模块，承载从 fp32 master 重建低精度权重的完整逻辑：MainCastContext 组装、MCore/HDO 两条 cast 路径、覆盖断言。

```
# miles/backends/megatron_utils/rematerialize_utils.py
"""Wiring for --rematerialize-param-from-master-weight. Rebuilds the low-precision
weights from the optimizer's master weights instead of a pinned CPU copy."""

import logging
from argparse import Namespace
from collections.abc import Callable, Iterator, Sequence

import torch

from miles.backends.megatron_utils.misc_utils import strip_param_name_prefix
from miles.utils.tensor_backper import MainCastContext

logger = logging.getLogger(__name__)


def _named_restore_extras(model: Sequence[torch.nn.Module]) -> Iterator[tuple[str, torch.
Tensor]]:
    """Tensors with no master weight to rebuild from, so they keep a pinned backup."""
    for vp_stage, model_module in enumerate(model):
        # expert_bias 是唯一被同步的 buffer，fp32 参数的 optimizer main 与参数本身别名，
        # 都无法从 fp32 master 独立重建，因此必须继续保留小的 pinned 备份。
        for name, buffer in model_module.named_buffers():
            if "expert_bias" in name:
                yield f"vp_stages.{vp_stage}.{strip_param_name_prefix(name)}", buffer
        for name, param in model_module.named_parameters():
            if param.dtype == torch.float32:
                yield f"vp_stages.{vp_stage}.{strip_param_name_prefix(name)}", param


def build_main_cast_context(
```

```
    args: Namespace, *, model: Sequence[torch.nn.Module], optimizer
) -> MainCastContext:
    extras = list(_named_restore_extras(model))
    extras_bytes = sum(t.numel() * t.element_size() for _, t in extras)
    logger.info(
        f"rematerialize-param-from-master-weight: {len(extras)} extra tensors "
        f"({extras_bytes / 2**20:.1f} MiB) kept in pinned backup: "
        f"{[name for name, _ in extras[:20]]}"
    )
    return MainCastContext(
        cast_main_to_params=_build_cast_main_to_params_fn(
            optimizer, precision_aware=args.use_precision_aware_optimizer
        ),
        model_chunks=model,
        extras_getter=lambda: _named_restore_extras(model),
        rematerializable_ids=_assert_rematerialize_coverage(model, extras),
        check=args.check_rematerialize_param_from_master_weight,
    )
```

miles/utils/tensor_backper.py

备份器核心抽象扩展，新增 MainCastContext 与 _TensorBackuperMainCast，是内存节省的直接载体。

```
class _TensorBackuperMainCast(TensorBackuper):
    """Rebuilds the actor weights instead of keeping a pinned CPU copy of them.

    Restore replays the step end's cast + all-gather, so it is bit-identical. Only
    `extras_getter` tensors keep a pinned backup. Non-actor tags (ref/teacher) have no
    master weights to rebuild from, so they keep full pinned copies via a delegated
    _TensorBackuperNormal.
    """

    _check_num_cycles = 2

    def __init__(self, source_getter, ctx: MainCastContext):
        super().__init__(source_getter=source_getter)
        self._ctx = ctx
        self._others = _TensorBackuperNormal(source_getter=source_getter)
        self._extras_backup: dict[str, torch.Tensor] = {}
        self._extras_backup_by_id: dict[int, torch.Tensor] = {}
        self._backup_count = 0
        self._expected_hashes: dict[str, str] | None = None

    @property
    def backup_tags(self):
        return ["actor", *self._others.backup_tags]

    @torch.no_grad()
    def backup(self, tag: str) -> None:
```

```
    # 非 actor 标签（ref/teacher）没有 master 可重建，委托普通 backuper 保持完整副本。
    if tag != "actor":
        return self._others.backup(tag)
    # 只备份无法从 master 重建的 extras（expert_bias、fp32 参数）。
    for name, tensor in self._ctx.extras_getter():
        if name not in self._extras_backup:
            self._extras_backup[name] = torch.empty_like(
                tensor, device=torch.device("cpu"), pin_memory=True
            )
        self._extras_backup[name].copy_(tensor.detach(), non_blocking=True)
        self._extras_backup_by_id[id(tensor)] = self._extras_backup[name]
    torch.cuda.synchronize()
    self._backup_count += 1
    # 仅前两个 cycle 做 SHA256 校验：cast 路径确定，且 param-gather 状态机的损坏
    # 会在下一步权重中暴露，故两个 cycle 足够捕获。
    if self._ctx.check and self._backup_count <= self._check_num_cycles:
        self._expected_hashes = self._compute_hashes()
    else:
        self._expected_hashes = None

@torch.no_grad()
def restore(self, tag: str) -> None:
    if tag != "actor":
        return self._others.restore(tag)
    # 用 master cast + param all-gather 重建本 rank 的 bf16 权重，与 step end 完全相同。
    self._ctx.cast_main_to_params()
    for model_chunk in self._ctx.model_chunks:
        model_chunk.start_param_sync(force_sync=True)
    # extras 仍从 pinned 备份恢复。
    for name, tensor in self._ctx.extras_getter():
        tensor.copy_(self._extras_backup[name], non_blocking=True)
    torch.cuda.synchronize()
    if self._expected_hashes is not None:
        self._verify_hashes()

def get(self, tag: str):
    if tag != "actor":
        return self._others.get(tag)
    # update_weights 期间 param buffer 处于 paused 区域，extras 只能从备份读取。
    out = {}
    for name, tensor in self._source_getter():
        backup = self._extras_backup_by_id.get(id(tensor))
        if backup is None:
            # 普通参数是 DDP buffer 成员，可以直接暴露 live GPU 指针。
            assert (
                id(tensor) in self._ctx.rematerializable_ids
            ), f"{name} is neither in the DDP param buffers nor in the extras backup"
            backup = tensor.detach()
        out[name] = backup
```

```
        return out
```

miles/utils/arguments.py

新增 CLI 开关与严谨的兼容性断言，防止不兼容配置在启动后期崩溃。

```
def _validate_rematerialize_param_from_master_weight(args):
    if not args.rematerialize_param_from_master_weight:
        return
    if args.debug_train_only:
        # update_weights 不会运行，param buffer 不会被 pause，直接静默关闭该优化。
        args.rematerialize_param_from_master_weight = False
        return
    # 先做后端 guard：后续断言读取的均为 megatron 专属参数，FSDP Namespace 会
    AttributeError。
    assert (
        args.train_backend == "megatron"
    ), "--rematerialize-param-from-master-weight reads Megatron's distributed-optimizer main
    params"
    from miles.backends.megatron_utils.lora_utils import is_lora_enabled

    assert not is_lora_enabled(args), "--rematerialize-param-from-master-weight does not
    support LoRA"
    assert not args.debug_disable_optimizer, "--debug-disable-optimizer leaves no main params
    to rematerialize from"
    assert not args.indep_dp, (
        "--rematerialize-param-from-master-weight drops the backup inside update_weights, "
        "which RayTrainGroup runs on the first alive cell only. Every other cell would "
        "keep it for the whole run. Lift this once all cells update weights."
    )
    assert args.colocate and args.offload_train
    assert args.offload_train_target == "cpu", (
        "--offload-train-target=disk streams the weights to NVMe and reads them back from "
        "GPU after resume, so there is no backup for the rebuild to replace"
    )
    # ... 其余断言省略
    args.disable_param_buffers_cpu_backup = True
    if args.ci_test:
        args.check_rematerialize_param_from_master_weight = True
```

评论区精华

review 中最有价值的交锋集中在正确性与分布式一致性：

- yushengsu-thu 指出 update_weights 中 pause(param_buffer) 不在 try/finally 里，一旦 update 块抛异常（引擎死亡、CI 校验失败）bf16 shard 将常驻；作者回应当前设计不可达但 fault-tolerance 下会有问题，故先以 --indep-dp assert 拒绝 FT 组合，后续再修。
- yushengsu-thu 指出 per-rank 的 SHA256 raise 在 DP>1 时会让其他 rank 冲进下一个集合通信并触发 NCCL watchdog hang，建议 all_reduce 失败标志；作者回应非 FT 场景 driver 直接崩溃，FT 场景被 indep_dp assert 挡住，未实现 all_reduce。

- gemini-code-assist[bot] 提出 requires_grad=False 的冻结参数不在 DDP buffer 中会触发覆盖断言，建议纳入 extras；最终代码未包含该分支。
- yushengsu-thu 指出参数校验读取 megatron 专属参数，FSDP 后端会 AttributeError，且 --debug-disable-optimizer 会让 optimizer=None 在 restore 时才暴露；作者在后续 commit 中"先 guard 后端"并把 debug-disable-optimizer 与 LoRA 提前到参数校验阶段。
- yushengsu-thu 还指出 tensor_backper 引入 backend 私有 helper 会让 megatron-less 环境 break；作者改为本地实现 _hash_tensor_sha256，11 个测试不再需要 importorskip。
- 冻结参数未纳入 extras 备份 (correctness): 最终代码未包含 requires_grad=False 分支；该建议未落地，后续若模型含冻结参数启用此 flag 会启动失败。
- 参数校验需先 guard 后端 (correctness): 作者在后续 commit 中先 guard 后端，并把 debug_disable_optimizer 与 LoRA 提前到参数校验阶段，已解决。
- pause(param_buffer) 的异常安全 (correctness): 作者回应"当前设计不可达，FT 会出问题"，以 indep_dp assert 拒绝 FT 组合，未改成 try/finally，风险被接受。
- SHA256 失败在 DP>1 下的分布式 hang (correctness): 作者回应非 FT 时 driver 直接崩溃，FT 被 indep_dp assert 挡住；all_reduce 方案未实现。
- utils 依赖 backend 私有 helper (design): 作者改为本地实现 _hash_tensor_sha256，不再依赖 ci_utils，11 个测试在无 megatron/sglang 环境通过，已解决。
- 为何 SHA256 只校验前 2 个 cycle (question): 作者解释：手动选择的值，cast 路径确定性高，且 param-gather 状态机损坏会在下一步权重中暴露，2 个 cycle 足够捕获陈旧 bug。
- weight-sync 窗口 GPU 峰值上升 (performance): 作者在 flag 的 help 文本中补充了该限制说明，已解决。
- 哈希函数可复用性 (design): 最终代码仍保留在 tensor_backper.py，未抽取复用；作者未明确回复。

风险与影响

- 风险：主要风险集中在启用该 flag 后的运行时：
  1. 权重重建正确性：依赖 DDP buffer 成员身份和 master cast 闭包；任何新加入 sync source 的 tensor 都会在 get() 时被断言挡住，但冻结参数（requires_grad=False）未被 extras 覆盖，遇到此类模型会在启动断言失败。
  2. 分布式失败模式：SHA256 校验的 per-rank raise 在 DP>1 下可能引发 NCCL hang；当前仅靠 indep_dp assert 隔离 FT，非 FT 场景 driver 崩溃但不会 hang。
  3. 异常安全缺口：update_weights 中 pause(param_buffer) 无 try/finally，异常路径下 bf16 shard 常驻，影响后续内存峰值。
  4. GPU 峰值上升：weight-sync 窗口内 param buffer 常驻使 GPU used 增加 per-rank bf16 shard（30B 上 +7.9 GB），help 文本已提示 80GB 显存卡需重调 mem-fraction。
  5. 兼容性面：参数校验拒绝 LoRA、disk offload、indep-dp、critic-only 等组合，误用会在启动早期清晰失败，属于可控风险。 - 影响：影响范围限定为 megatron 后端的 colocate + offload-train(cpu) 训练：启用后单节点 host 峰值内存降低 8.5%-18%（GLM-5.2 744B 上 -129 GB/ 节点），step 时间变化约 +1.4 s 到 -1.5 s（稳态 1000 s 量级），权重 bit 级一致。对非 colocate、FSDP、LoRA、disk offload、

fault-tolerance 用户无影响（参数校验直接拒绝）。对团队而言，这是 colocate 大模型训练 host 内存瓶颈的关键缓解手段，尤其适合 Grace GPU 与 744B 级 MoE 模型。 - 风险标记：核心生命周期变更，启用需严格参数组合，SHA256 失败可能造成分布式 hang，冻结参数未被覆盖，异常安全未完全解决

关联脉络

- PR #1588 critic-only warmup: remove redundant update_weights（PR body 引用）：PR body 明确提到 critic-only warmup 步骤会被 #1588 移除冗余更新并解除本 PR 的 num_critic_only_steps fence。
- PR #2203 Revert "Reject --disable-weights-backuper for LoRA + colocate + offload-train" (#2077): 同属 weights_backuper 参数行为与 arguments.py 校验，且本 PR 要求 enable_weights_backuper=True，与 #2077/#2203 的禁用断言直接相关。
- PR #2077 Reject --disable-weights-backuper for LoRA + colocate + offload-train: 同样在 arguments.py 中为 backuper 组合引入严格校验，与本 PR 的兼容性断言形成同一参数校验脉络。
- PR #2224 fix: derive --critic-save from --save so PPO critic checkpoints are not silently skipped: 同为 arguments.py 的默认值 / 校验改动，围绕 PPO 检查点与参数一致性，体现该文件是高频变更汇聚点。