PR #6555 完整报告

verl-project/verl

[megatron] feat: add dynamic context parallel scheduling

合并时间：2026-08-11 13:13

原文链接：http://prhub.com.cn/verl-project/verl/pull/6555

执行摘要

- 一句话：Megatron 新增动态 CP 调度，长尾序列吞吐提升约 60%
- 推荐动作：值得精读。本 PR 的核心价值有三：一是 thin adapter 设计——把调度算法委托给上游 Megatron-Core，verl 只保留数据路由与输出收集，避免重复实现带来的正确性风险（早期自研 1470 行调度器因多个逻辑错误被整体移除）；二是集中式 feature 校验 _check_dcp_unsupported_features，把所有不兼容组合收敛到单一入口；三是 DCP 下 local token loss 与输出顺序恢复的实现细节（leader 收集 + all_gather_object + jagged 重建），可作为分布式训练数据流设计的参考。建议重点关注 verl/utils/dynamic_cp_scheduler.py 与 verl/workers/engine/megatron/transformer_impl.py 两处。

功能与动机

PR body 明确指出，静态 CP 在长尾序列长度分布（1K 到 16K）下会造成大量不必要的 CP 通信与 padding 工作，而 DCP 可以在每个 packed micro-batch 上按需选用更小的本地 CP group，从而减少开销。ISEEKYAN 在 review 中要求提供性能与精度曲线证明有效性（"it would be better if you could provide some perf and precision curve to prove the effectiveness of DCP"），作者随后补充了 +60.6% 吞吐与 loss 几乎不变的基准。此外，ISEEKYAN 强调不能为了 Megatron 特性改动通用 loss 函数（"it is not permitted to modify the loss function for megatron's specific feature... we must make our DCP free of changing losses"），这成为实现中把裁剪逻辑放在路由侧而非 loss 侧的重要约束。

实现拆解

整体实现分为 5 步：

1. 新增调度适配层 verl/utils/dynamic_cp_scheduler.py（新增 286 行）：get_megatron_dynamic_cp_scheduler_cls() 从 megatron.core.datasets.data_schedule 导入 DefaultDynamicCPScheduler，若 Megatron-Core 不含 NVIDIA/Megatron-LM#5154 则直接报错；DynamicCPScheduler.schedule() 按 DPxCP group 与 max_seqlen_per_dp_cp_rank 计算每个 micro-batch 的本地 CP group，并通过 tu.assign_non_tensor_data 给 micro-batch 附加 local_cp_size、_dcp_sample_ids、_dcp_group_leader、_dcp_padding_mask 等元数据；postprocess_dynamic_cp_batch() 只收集每个 CP group leader 的输出，经 all_gather_object 汇总后按 sample id 恢复原始顺序并重建 jagged NestedTensor。

2. 引擎集成 verl/workers/engine/megatron/transformer_impl.py（+254/-95）：_init_device_mesh() 在 mpu 已初始化时校验 DPxCP 世界大小（偶数且不小于 2）并尝试获取动态 DPxCP 分组，未初始化时向 initialize_model_parallel 传入 dynamic_context_parallel=True；_check_dcp_unsupported_features() 集中拒绝 FP8、VPP、value model、多模态、distillation 与 moe_router_fusion；fused forward 通过 _resolve_fused_temperature() 支持标量 / 均匀温度；_attach_dcp_recorded_routes() 按 sample id 把记录的 topk 路由回挂到各 micro-batch 的 model_output。
3. 配套数据契约改造：verl/utils/megatron/router_replay_utils.py 中 _context_parallel_layout() 抽取共享，merge_router_topk_indices()/set_router_replay_data() 增加 local_cp_size 与 cp_layout 透传，新增 align_r3_router_replay_data() 为 rollout 缺失的最后一行 route 补占位，pp_gather() 对嵌套路由先搬到 CPU 再 all_gather；verl/models/mcore/mtp_patch.py 新增 _resolve_cp_group()（优先使用 packed_seq_params.cp_group）与 _get_mtp_loss_config()（按 token 归一化因子调整 MTP scaling）；verl/utils/megatron_utils.py 删除旧版 dynamic_cp_split_batch/dynamic_cp_merge_output（基于 2 的幂切分的简化实现，-82 行）。
4. 配置与文档：verl/trainer/config/engine/megatron.yaml、ref/megatron_ref.yaml、_generated_ppo_megatron_trainer.yaml 新增 dynamic_context_parallel 相关配置；新增 docs/advance/dynamic_context_parallel.rst 说明用法与限制；PR body 附带 Qwen3-30B-A3B 的吞吐与 loss 对比基准。
5. 测试配套（6 个文件）：新增 tests/utils/test_dynamic_cp_scheduler.py（调度、padding mask、输出顺序恢复、引擎契约、fused forward 传参）、tests/utils/test_megatron_router_replay_dcp.py（R2/R3 路由与动态 CP group）、tests/special_distributed/test_megatron_dynamic_cp_features.py（真实 4 rank 下 CP2+CP1 混合的 router replay 与 MTP 前反向）、tests/utils/test_megatron_mtp_dcp.py（MTP loss 动态 group 与 token 归一化）；修改 tests/utils/test_megatron_bshd_preprocess.py（多维 THD 序列维 padding）与 tests/special_distributed/run_all.sh。

关键文件：

- verl/utils/dynamic_cp_scheduler.py（模块 调度适配；类别 source；类型 core-logic；符号 get_megatron_dynamic_cp_scheduler_cls, _local_padding_mask, _cp_members, DynamicCPScheduler）：DCP 核心新增文件：Megatron-Core 调度器的 thin TensorDict adapter，负责 micro-batch 排程、padding mask 构造与输出收集恢复顺序，是整个功能的枢纽。
- verl/workers/engine/megatron/transformer_impl.py（模块 引擎集成；类别 source；类型 core-logic；符号 _resolve_fused_temperature, _validate_dcp_world_size, _check_dcp_unsupported_features, _attach_dcp_recorded_routes）：DCP 的引擎集成入口：设备网格初始化、不支持特性集中校验、fused forward 温度解析与 router replay 路由回挂，静态 CP 用户也受此重构影响。
- verl/utils/megatron/router_replay_utils.py（模块 路由回放；类别 source；类型 core-logic；符号 _context_parallel_layout, get_num_layers_to_build, merge_router_topk_indices, align_r3_router_replay_data）：router replay（R2/R3）在

DCP 下的数据契约改造：local_cp_size 透传、R3 路由对齐与 PP 跨 rank 收集的 device 归一化。

- verl/utils/megatron_utils.py（模块 训练工具；类别 source；类型 core-logic；符号 dynamic_cp_split_batch, dynamic_cp_merge_output, get_megatron_mtp_loss）：删除旧版基于 2 的幂切分的 dynamic_cp_split_batch/dynamic_cp_merge_output（-82 行），统一到新调度方案，并调整 MTP loss 归一化逻辑以兼容新版 MCore tracker。
- verl/models/mcore/mtp_patch.py（模块 MTP 支持；类别 source；类型 data-contract；符号 _resolve_cp_group, _get_mtp_loss_config）：MTP 在 DCP 下的数据契约：每个 micro-batch 使用 packed_seq_params 携带的动态 CP group 做 roll 与 loss 通信，并按 token 归一化因子调整 scaling。
- verl/models/mcore/util.py（模块 序列预处理；类别 source；类型 data-contract；符号 _packed_seq_params_supports, preprocess_thd_engine, postprocess_thd_engine）：THD 预处理与后处理的核心工具：支持 cp_partition_mode 透传，修正多维张量（如 topk logits）padding 只作用于序列维。
- tests/utils/test_dynamic_cp_scheduler.py（模块 单元测试；类别 test；类型 test-coverage；符号 _FakeGroup, _nested, _batch, test_schedule_adapts_mcore_assignments）：新增的调度器单元测试：覆盖 schedule 的 rank 分配、padding mask 真实 token 计数、非法容量与未对齐 group 拒绝、引擎 DCP 契约、fused forward 参数透传与输出顺序恢复。
- tests/special_distributed/test_megatron_dynamic_cp_features.py（模块 分布式测试；类别 test；类型 test-coverage；符号 _model_parallel, _sample, _route, test_router_replay_uses_each_microbatch_cp_group）：真实 4 rank 分布式测试：验证 CP2 与独立 CP1 micro-batch 混合场景下的 router replay 与 MTP roll/backward，是 DCP 端到端正确性的关键保障。
- docs/advance/dynamic_context_parallel.rst（模块 文档；类别 docs；类型 documentation）：新增 DCP 使用文档：说明开启方式、与静态 CP 的关系、适用场景与限制，是用户入口。
- verl/trainer/config/engine/megatron.yaml（模块 配置；类别 config；类型 configuration）：Megatron 引擎配置新增 dynamic_context_parallel 相关字段，是功能开关的入口配置。

关键符号：DynamicCPScheduler.schedule, postprocess_dynamic_cp_batch, _local_padding_mask, _cp_members, _check_dcp_unsupported_features, _attach_dcp_recorded_routes, _resolve_fused_temperature, _resolve_cp_group, _get_mtp_loss_config, align_r3_router_replay_data, merge_router_topk_indices, set_router_replay_data, preprocess_thd_engine, postprocess_thd_engine

关键源码片段

verl/workers/engine/megatron/transformer_impl.py

DCP 的引擎集成入口：设备网格初始化、不支持特性集中校验、fused forward 温度解析与 router replay 路由回挂，静态 CP 用户也受此重构影响。

```
# DCP 能力集中校验：所有不兼容组合都收敛到这里，避免散落在各函数中
```

```
def _check_dcp_unsupported_features(engine_config, model_config, tf_config=None, batch=
None) -> None:
    """随着模型、transformer 与 batch 状态逐步可得，分阶段校验 DCP 能力边界。"""
    if not engine_config.dynamic_context_parallel:
        return
    # remove padding 是 THD 打包的前提
    if not engine_config.use_remove_padding:
        raise ValueError("dynamic_context_parallel requires use_remove_padding=True")
    if model_config.model_type == "value_model":
        raise NotImplementedError("Dynamic CP currently supports language models only")
    if hasattr(model_config.hf_config, "vision_config"):
        raise NotImplementedError("Dynamic CP does not support multimodal models")
    if engine_config.virtual_pipeline_model_parallel_size not in (None, 1):
        raise NotImplementedError("Dynamic CP does not support virtual pipeline parallelism")

    if tf_config is not None:
        if getattr(tf_config, "fp8", None) not in (None, False):
            raise NotImplementedError("Dynamic CP does not support FP8 training")
        # moe_router_fusion 会绕过 replay hooks，所以与 router replay 互斥
        if engine_config.router_replay.mode != "disabled" and getattr(tf_config, "moe_router_
        fusion", False):
            raise NotImplementedError("Dynamic CP router replay requires moe_router_fusion=
            False")

    if batch is not None:
        if tu.get_non_tensor_data(batch, key="distillation_use_topk", default=False) or tu.get_non_
        tensor_data(
            batch, key="distillation_only", default=False
        ):
            raise NotImplementedError("Dynamic CP does not support distillation")


def _attach_dcp_recorded_routes(losses_reduced: list[dict], layers_topk_idx: torch.Tensor) ->
None:
    """把 router replay 记录的 topk 路由按 DCP micro-batch 的 sample id 回挂。

    不能丢样本也不能虚构样本，因此前后都要做数量核对。
    """
    recorded_routes = list(layers_topk_idx.unbind())
    cursor = 0
    for micro_batch_index, micro_output in enumerate(losses_reduced):
        micro_sample_ids = micro_output.get(DCP_SAMPLE_IDS)
        if not micro_sample_ids:
            raise RuntimeError(f"DCP router replay micro-batch {micro_batch_index} has no sample
            ids")
        end = cursor + len(micro_sample_ids)
        if end > len(recorded_routes):
            raise RuntimeError(
                "DCP router replay produced fewer route tensors than scheduled samples: "
```

```
                f"need {end}, got {len(recorded_routes)}"
            )
        # 组装成 jagged NestedTensor，保持与 model_output 其他字段一致的布局
        micro_output.setdefault("model_output", {})["routed_experts"] = torch.nested.as_nested_
        tensor(
            recorded_routes[cursor:end], layout=torch.jagged
        )
        cursor = end

    if cursor != len(recorded_routes):
        raise RuntimeError(
            f"DCP router replay produced unconsumed route tensors: consumed {cursor}, recorded
            {len(recorded_routes)}"
        )
```

verl/utils/megatron/router_replay_utils.py

router replay（R2/R3）在 DCP 下的数据契约改造：local_cp_size 透传、R3 路由对齐与 PP 跨 rank 收集的 device 归一化。

```
def align_r3_router_replay_data(layers_topk_idx: torch.Tensor, input_ids: torch.Tensor) -> torch.
Tensor:
    """把 rollout 阶段捕获的 R3 路由与完整训练输入逐序列对齐。

    自回归 rollout 只为每个生成 token 记录路由，因此最后一个生成 token
    没有对应 route；R3 replay mask 会把最后的模型行保留为 native，
    这里为每个受影响序列补一个被忽略的占位行，保证路由与模型张量形状一致。
    """
    if not layers_topk_idx.is_nested or not input_ids.is_nested:
        raise TypeError("R3 router replay requires jagged route targets and input_ids")

    route_parts = list(layers_topk_idx.unbind())
    input_lens = [int(x) for x in input_ids.offsets().diff().tolist()]
    if len(route_parts) != len(input_lens):
        raise RuntimeError(
            f"R3 router replay has {len(route_parts)} route sequences for {len(input_lens)} input
            sequences"
        )

    aligned_parts = []
    for sample_id, (routes, input_len) in enumerate(zip(route_parts, input_lens, strict=True)):
        route_len = routes.shape[0]
        if route_len == input_len:
            # 已经对齐，直接使用
            aligned_parts.append(routes)
        elif route_len == input_len - 1:
            # 缺最后一行 route，补一个零占位，保持形状完全一致
            placeholder = torch.zeros((1, *routes.shape[1:]), dtype=routes.dtype, device=routes.
            device)
            aligned_parts.append(torch.cat((routes, placeholder), dim=0))
```

```
            else:
                raise RuntimeError(
                    f"R3 router replay sample {sample_id} has {route_len} route rows for {input_len}
                    input tokens; "
                    "expected equal lengths or exactly one missing final route"
                )

    return torch.nested.as_nested_tensor(aligned_parts, layout=torch.jagged)
```

评论区精华

核心讨论集中在三处：

1. 是否应该复用 Megatron-Core 调度器：ISEEKYAN 在 verl/utils/dynamic_cp_scheduler.py 上提问 "is it possible that we import the core algorithm from megatron-core instead of rewrite on verl?"，作者回复已改为导入 DefaultDynamicCPScheduler，verl 只保留数据路由（data routing）。这一决策同时消除了 gemini-code-assist 指出的自研调度器多个问题（dcp_make_buckets_equal 阈值递减过快、align_sample_id_groups 切片赋值改变列表长度、fill_empty_gpus 低估打包 workload）。
2. 不得修改通用 loss 函数：ISEEKYAN 在 verl/workers/utils/losses.py 上明确拒绝为 Megatron 特性改动通用且后端无关的 loss，作者回复 "got it"；最终版本中 losses.py 无 DCP 改动，序列裁剪改由路由侧完成。
3. P0 兼容清单：ISEEKYAN 问 fused kernel（线性交叉熵）对长上下文 RL 至关重要，建议按 fused_kernel、router replay、mtp 排序兼容；作者回复最新修订已支持全部三项，并说明 moe_router_fusion 仍显式不支持（绕过 replay hooks）。

- 是否复用 Megatron-Core 调度算法而非重写 (design): 核心排程委托给 Megatron-Core 的 DefaultDynamicCPScheduler，verl 侧仅做 TensorDict 适配、padding mask 构造与输出收集；同时消除了自研实现中多个逻辑缺陷。
- 不得为 Megatron 特性修改通用 loss 函数 (design): 最终版本中 losses.py 无 DCP 相关改动，序列长度差异改由路由侧在 micro-batch 准备阶段处理。
- fused kernel / router replay / MTP 兼容优先级 (performance): 三项均兼容：fused CE 接收 local_cp_size 与 DCP padding mask；R2/R3 使用动态 CP group；MTP 使用 packed_seq_params 的 cp_group。moe_router_fusion 被 _check_dcp_unsupported_features 拒绝。
- 集中管理不支持的 feature (design): 新增 _check_dcp_unsupported_features 函数，集中校验 FP8、VPP、value model、多模态、distillation、moe_router_fusion。
- 自研调度器 dcp_make_buckets_equal 逻辑错误 (correctness): 该自研调度器在后续重构中被整体移除，核心排程改由 Megatron-Core 承担，问题随代码删除而消失。
- NestedTensor 上直接调用 sum() 的兼容性 (correctness): 作者回复 fixed，改用 data.get('loss_mask') 并在缺失时回退到 1.0，保证 loss 归一化在 DCP 路径下可用。
- 要求提供性能与精度曲线 (question): 作者在 PR body 中补充 Qwen3-30B-A3B SFT 基准：吞吐 +60.6%，loss 相对差异 0.0229%，并附曲线图。

- DCP 处理是否应移入 prepare_micro_batches (design): DCP 排程逻辑被组织在 forward_backward_batch 的 dcp 分支中，并通过 DynamicCPScheduler 与 postprocess_dynamic_cp_batch 封装，保持主体流程简洁。

风险与影响

- 风险：具体风险如下：
- 上游依赖硬性门槛：get_megatron_dynamic_cp_scheduler_cls() 要求 Megatron-Core 包含 NVIDIA/Megatron-LM#5154（commit d2e7ec5b），版本不满足时在引擎初始化阶段直接 RuntimeError；若 actor/ref 等共置引擎的 Megatron 初始化不一致，会因动态 DPxCP group 缺失而报错（代码已提供显式诊断）。
- 核心路径回归：transformer_impl.py 改动幅度大（+254/-95），覆盖 _init_device_mesh、_build_tf_config、forward_backward_batch 等关键路径；即使 dynamic_context_parallel=False，重构后的初始化逻辑与 context_parallel_size 设置仍可能影响静态 CP 用户。
- NestedTensor 兼容性：review 曾指出对 jagged 布局直接调用 .sum() 在部分 PyTorch 版本不可靠；当前实现改为先取 loss_mask 再聚合，但 postprocess_dynamic_cp_batch 中 torch.nested.as_nested_tensor 的可用性仍依赖 PyTorch 版本。
- 输出收集开销：postprocess_dynamic_cp_batch 通过 all_gather_object 把每个 leader 的 model_output 先 detach 到 CPU 再收集，长序列大 log_prob 场景下存在额外同步与拷贝开销。
- R3 对齐假设：align_r3_router_replay_data 假设 route 行数等于 input_len 或 input_len-1，否则抛错；rollout 侧若改变路由记录粒度，该假设会静默失效前的报错会中断训练。
- 功能限制面：FP8、VPP、value model、多模态、distillation、moe_router_fusion 均被显式拒绝，用户误用时错误信息清晰，但覆盖范围有限。
- 影响：对用户：开启 actor_rollout_ref.actor.megatron.dynamic_context_parallel=True 后，长尾序列分布下的 SFT/RL 吞吐显著提升（Qwen3-30B-A3B 验证 +60.6%）、loss 几乎不变（相对差异 0.0229%）；但需要同步升级 Megatron-Core 到含 #5154 的版本，且 DCP 暂不支持 FP8/VPP/ 多模态等组合。对系统：micro-batch 数据契约扩展（local_cp_size、_dcp_sample_ids、_dcp_padding_mask 等），router replay、MTP、fused forward 三个子系统均需感知动态 CP group，后续维护面扩大；megatron_utils.py 中旧的分批 / 合并逻辑被删除，依赖旧 API 的代码需迁移。对团队：verl 与 Megatron-Core 的耦合收紧，mcore 上游 API（PackedSeqParams 字段、DefaultDynamicCPScheduler 接口）变化会直接影响 DCP 路径，CI 中新增分布式测试也提升了回归保障门槛。
- 风险标记：依赖 Megatron-Core 上游版本，核心路径变更，限制 FP8/VPP/ 多模态，NestedTensor 兼容性，R3 对齐假设，输出收集同步开销

关联脉络

- PR #7297 [megatron] fix: make DeepSeek-V4 context parallelism actually runnable: 与本 PR 同属 Megatron CP 功能线，均修改 verl/utils/megatron/router_replay_utils.py 与

verl/models/mcore/util.py，DCP 需要与其保持 CP 布局（zigzag/contiguous）传递的一致性。

- PR #7261 [megatron] fix: pad multidimensional THD tensors along the sequence dimension: 与本 PR 对 verl/models/mcore/util.py 的多维 THD padding 修复高度相关，两者共同保证 topk logits 等多维张量在 CP 下不被错误 pad。
- PR #7328 [megatron] fix: forward mhc_multistream to MTP and skip activation reclaim for MTP checkpoints: 同样改动 verl/models/mcore/mtp_patch.py，与 DCP 的 MTP 支持在同一个 MTP patch 区域演进，需避免冲突并共同维护 MTP 前向兼容性。
- PR #6933 [megatron] feat: migrate fused logprob/entropy from GPTModel.forward monkey-patch to Megatron output_processor hook: 改动了 verl/models/mcore/model_forward_fused.py，本 PR 也为 fused forward 增加 local_cp_size 与 router padding mask 透传，两条改动共同决定 fused kernel 在 DCP 下的行为。