PR #6901 完整报告

verl-project/verl

[megatron, perf] feat: pad BSHD micro-batches to mini-batch max seq_len

合并时间：2026-07-02 10:06

原文链接：http://prhub.com.cn/verl-project/verl/pull/6901

执行摘要

- 一句话：Megatron BSHD 统一 micro-batch 序列长度避免 cuDNN 重复编译
- 推荐动作：值得精读。该 PR 精准定位了 cuDNN Graph Build 与 micro-batch 填充策略的交互问题，设计简洁有效。建议关注空 batch 安全性的后续修正，并考虑将此类优化扩展至更通用的 THD 路径。

功能与动机

cuDNN fused-attention 针对每个不同的 s_q 形状会触发一次约 1s 的 Graph Build（CPU 同步），导致训练吞吐降低。该问题在 NVIDIA/TransformerEngine#2033 中有相关讨论。此 PR 通过统一 micro-batch 填充长度来避免重复编译，提升 BSHD 路径性能。

实现拆解

1. 配置层：在 McoreEngineConfig 中添加 pad_bshd_to_minibatch_max: bool = True（verl/workers/config/engine.py），并同步到 megatron.yaml 和 _generated_ppo_megatron_trainer.yaml。
2. 引擎核心：在 MegatronEngineWithLMHead.forward_backward_batch 中，当标志启用且为 BSHD 路径时，计算 mini-batch 全局最大序列长度（raw，未对齐），并通过 NonTensorData 注入到每个 micro-batch（verl/workers/engine/megatron/transformer_impl.py）。
3. 数据预处理：preprocess_bshd_engine 新增 forced_max_seqlen 参数，当传入时覆盖 seqlens_in_batch.max()；所有 TP/CP/FP8 对齐逻辑保持不变（verl/models/mcore/util.py）。
4. 前向传播：gptmodel_forward_model_engine 新增 forced_max_seqlen 参数，并传递到所有 BSHD 预处理调用（包括 MTP label/loss_mask、logits_processor 参数）以及 VLM 的 build_vlm_attn_mask_bshd 调用，确保 VLM 分支内 logits.shape[:2] 与 label.shape[:2] 一致（verl/models/mcore/model_forward.py）。
5. 配置同步：更新 megatron.yaml 和 _generated_ppo_megatron_trainer.yaml，添加注释并默认启用。

关键文件：

- verl/models/mcore/util.py（模块 预处理；类别 source；类型 data-contract；符号 preprocess_bshd_engine, build_vlm_attn_mask_bshd）：核心数据预处理函数

preprocess_bshd_engine 和 build_vlm_attn_mask_bshd 新增 forced_max_seqlen 参数，支持统一填充长度。

- verl/models/mcore/model_forward.py（模块 前向传播；类别 source；类型 data-contract；符号 gptmodel_forward_model_engine）：前向函数 gptmodel_forward_model_engine 新增 forced_max_seqlen 参数，并传递给所有 BSHD 预处理调用及 VLM 掩码构建，确保形状一致。
- verl/workers/engine/megatron/transformer_impl.py（模块 引擎核心；类别 source；类型 core-logic；符号 forward_backward_batch, forward_step, logits_processor）：引擎核心 forward_backward_batch 计算 mini-batch 全局最大序列长度并通过 NonTensorData 注入每个 micro-batch；forward_step 读取并传递该值。
- verl/workers/config/engine.py（模块 引擎配置；类别 source；类型 core-logic）：在 McoreEngineConfig 中添加 pad_bshd_to_minibatch_max: bool = True 配置项，控制是否启用统一填充。
- verl/trainer/config/engine/megatron.yaml（模块 训练配置；类别 config；类型 configuration）：配置模板，添加 pad_bshd_to_minibatch_max: True 并附注释。
- verl/trainer/config/_generated_ppo_megatron_trainer.yaml（模块 训练配置；类别 config；类型 configuration）：自动生成的训练配置，同步新增 pad_bshd_to_minibatch_max 条目（actor、ref、critic 三段）。

关键符号：preprocess_bshd_engine, build_vlm_attn_mask_bshd, gptmodel_forward_model_engine, forward_backward_batch, forward_step, logits_processor

关键源码片段

verl/models/mcore/util.py

核心数据预处理函数 preprocess_bshd_engine 和 build_vlm_attn_mask_bshd 新增 forced_max_seqlen 参数，支持统一填充长度。

```
def preprocess_bshd_engine(
    input_ids: torch.Tensor,
    pre_process: bool = True,
    need_roll: bool = False,
    use_fp8_padding: bool = False,
    forced_max_seqlen: Optional[int] = None,
):
    """
    Preprocess bshd sequences
    return input_ids, attention_mask, position_ids

    The input is a jagged nested tensor with shape [batch, seq, ...]. Any
    dense dimensions after seq are preserved in the returned padded tensor.

    When forced_max_seqlen is given, it overrides the per-micro-batch
    seqlens_in_batch.max() as the raw padding target. Callers that want to
    align tensor shapes across micro-batches (e.g. to share a cuDNN fused-attention
```

```
    execution plan) pass the *mini-batch* global max here; any TP/CP/FP8 alignment
    is still applied below, so this argument should be the unaligned raw max.
    """
    cp_size = mpu.get_context_parallel_world_size()
    cp_rank = mpu.get_context_parallel_rank()

    batch_size = input_ids.shape[0]
    dense_shape = tuple(input_ids.shape[2:])
    seqlens_in_batch = input_ids.offsets().diff()
    # 关键变更：当 forced_max_seqlen 不为 None 时，使用全局最大值而非 micro-batch 自身最大值
    max_seqlen = forced_max_seqlen if forced_max_seqlen is not None else seqlens_in_batch.
    max().item()

    tp_size = mpu.get_tensor_model_parallel_world_size()
    # TP/CP 对齐逻辑保持不变，基于 max_seqlen 计算最终的 padding 长度
    align_size = tp_size * cp_size * 2 if cp_size > 1 else tp_size
    if align_size > 1:
        pad_size = (align_size - max_seqlen % align_size) % align_size
        max_seqlen += pad_size
    if use_fp8_padding:
        # FP8 块量化所需的额外对齐
        fp8_total_align = 128 * tp_size * cp_size
        fp8_seq_align = fp8_total_align // math.gcd(batch_size, fp8_total_align)
        fp8_seq_align_ = math.lcm(fp8_seq_align, align_size)
        max_seqlen = ((max_seqlen + fp8_seq_align_ - 1) // fp8_seq_align_) * fp8_seq_align_

    local_max_seqlen = max_seqlen // cp_size if cp_size > 1 else max_seqlen
    # 后续 padding 和 mask 创建使用统一的 local_max_seqlen，确保形状一致
    attention_mask = torch.zeros(batch_size, local_max_seqlen, dtype=torch.bool, device=input_
    ids.device)
    input_ids_bshd = torch.zeros(
        (batch_size, local_max_seqlen, *dense_shape), dtype=input_ids.dtype, device=input_ids.
        device
    )
    # ... 填充循环，基于 seqlens_in_batch 复制数据
```

verl/workers/engine/megatron/transformer_impl.py

引擎核心 forward_backward_batch 计算 mini-batch 全局最大序列长度并通过 NonTensorData 注入每个 micro-batch；forward_step 读取并传递该值。

```
  def forward_backward_batch(self, data: TensorDict, loss_function: Callable, forward_only=False)
  -> Any:
      tu.assign_non_tensor(data, sp_size=self.engine_config.context_parallel_size)
      # ... 计算 batch_num_tokens 等

      # BSHD path only: pad every micro-batch to the mini-batch's global max seq_len so the
      # padded s_q is shared -> cuDNN plan built once per shape. Raw (unaligned)
      # max; TP/CP/FP8 alignment is applied inside preprocess_bshd_engine.
      pad_bshd_to_minibatch_max = self.engine_config.pad_bshd_to_minibatch_max
```

```
global_max_seqlen = None
if pad_bshd_to_minibatch_max and not self.engine_config.use_remove_padding and "input_
ids" in data:
    input_ids_for_max = data["input_ids"]
    if input_ids_for_max.is_nested:
        # 计算整个 mini-batch 的 unaligned 全局最大序列长度
        global_max_seqlen = int(input_ids_for_max.offsets().diff().max().item())

vpp_size = mpu.get_virtual_pipeline_model_parallel_world_size()
# ... prepare_micro_batches

# 将 global_max_seqlen 注入每个 micro-batch
for micro_batch in micro_batches:
    tu.assign_non_tensor(micro_batch, num_micro_batch=n_micro_batch)
    if global_max_seqlen is not None:
        tu.assign_non_tensor(micro_batch, forced_max_seqlen=global_max_seqlen)

# 后续 forward_step 中通过 tu.get_non_tensor_data 读取 forced_max_seqlen 并传给模型前向
# 例如：forced_max_seqlen=tu.get_non_tensor_data(data=batch, key="forced_max_seqlen",
default=None)
```

评论区精华

- 空 batch 安全问题：gemini-code-assist[bot] 在 transformer_impl.py 和 util.py 中指出，当某些 rank 在分布式训练中收到空 batch 时，.max() 会引发 RuntimeError，建议添加 guard。这些建议未在本次合并中采纳，但值得后续跟进。
- 默认启用：wuxibin89 询问是否应默认启用。YaoweiFan 改为默认 true，并说明对 THD 路径无影响。
  - 空 batch 安全性 (correctness): 未采纳，但建议关注后续修复。
  - 默认启用决策 (design): 改为默认 true。

风险与影响

- 风险：
  - 空 batch 崩溃风险：在动态 batch 或序列并行下，某些 rank 可能收到空批次，此时 .offsets().diff().max() 会报错（已在 review 中指出来自未修复）。
  - VLM 兼容性风险：VLM 分支若未正确处理 forced_max_seqlen，会导致 logits 与 label 形状不匹配；PR 已在 build_vlm_attn_mask_bshd 和 model_forward.py 中统一传递。
  - 性能倒退风险：对已使用 use_length_grouped_bsz 的工作负载，额外 padding 可能增加计算量，但 cuDNN 计划重用带来的收益通常更大。
- 影响：
  - 用户：BSHD 路径用户默认获得性能提升（~25% 总步时缩短），无需改动配置。THD 路径不受影响。
  - 系统：减少 cuDNN 编译阶段 CPU 同步开销，提升 GPU 利用率。

- 团队：需关注空 batch 安全性的后续修复，以及未来可能向 THD 路径或 FSDP 推广类似优化。
- 风险标记：空 batch 潜在崩溃，默认启用可能引入意外行为，VLM 路径已修复但需验证

关联脉络

- PR #5338 [trainer] use_length_grouped_bsz for FSDP: 针对 FSDP 通过长度排序减少 padding 浪费，与当前 PR 针对 Megatron BSHD 路径的 cuDNN 计划优化互补。
- PR #6506 [model] preserve BSHD top-k trailing dims: 与当前 PR 同属 BSHD 路径改进，但修改位置不同，无冲突。