PR #51434 完整报告

vllm-project/vllm

[Perf] Optimize DeepSeek V3.2 sequence parallelism

合并时间：2026-08-08 07:51

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51434

执行摘要

- 一句话：DSv3.2 序列并行全链路化，吞吐提升 3.6%~5.2%
- 推荐动作：值得精读。三个设计决策最值得关注：(1) dense MLP 在 SP 下复制以换取通信消除的显存 / 性能权衡阈值；(2) 从手工 pad + TP collective 收敛到 sp_all_gather / sp_reduce_scatter / sp_shard 统一原语，对其它模型的迁移价值；(3) MTP 输入 shard 点前置、输出端打包一次性 gather 的放置原则。建议阅读时对比 tests/models/deepseek_v32/test_sequence_parallel.py 的 token 数契约断言，理解数据流的 full/shard 边界。

功能与动机

原实现只在 MoE 层启用序列并行，导致 dense/MoE 边界反复在 full 与 sharded 状态之间转换，使用原始 tensor-parallel 集合通信并物化 residual 状态。PR body 明确目标："keep DeepSeek V3.2 hidden and residual states sequence-sharded across every decoder layer, including the dense prefix"，并复用通用优化的 sequence-parallel gather/reduce-scatter 辅助函数，使 dense MLP 在 SP 下复制到各 rank，与 Kimi K3 和 DeepSeek V4 的数据流对齐，从而移除转换开销并压缩 CUDA graph 内存。

实现拆解

实现按 5 步展开：

1. 模型入口一次性 shard（vllm/models/deepseek_v32/nvidia/model.py）：DeepseekV32Model.__init__ 新增与 decoder 层一致的 use_sequence_parallel 判定（use_sequence_parallel_moe and pipeline_parallel_size == 1）；forward 在进入 layer 循环前通过 sp_shard 将 hidden_states 切为本地 token shard，并用 sp_padding_mask 同步维护 VLLM_MOE_SKIP_PADDING 下的 padding 掩码，同时断言 residual is None（SP 与 PP 不兼容）。原 _all_gather_sp_states 辅助函数及 dense/MoE 边界按序列长度判断的逐层 gather 被整体删除。
2. Decoder 层全程保持 shard：DeepseekV32DecoderLayer 用整层 use_sequence_parallel 取代仅针对 MoE 的 use_sequence_parallel_moe；dense MLP（DeepseekV2MLP）构造传入 is_sequence_parallel=True 实现每 rank 复制，MoE 分支保持 already_sequence_parallel=True 执行。attention 前后的通信从手工 pad + tensor_model_parallel_all_gather/reduce_scatter 替换为 sp_all_gather / sp_reduce_scatter；residual 不再需要 sequence_parallel_chunk 切分，因为其在 SP 下

始终是 shard。

3. aux hidden states 打包一次性 gather：aux_hidden_state_layers 的辅助状态以 shard 形式就地收集（hidden_states + residual），不再逐层 gather；模型出口处与最终 hidden 经torch.cat按hidden_size维度打包后只执行一次sp_all_gather，再切回各分量。
4. MTP 链路 shard 前置（vllm/models/deepseek_v32/nvidia/mtp.py）：DeepseekV32MultiTokenPredictorLayer.forward 在 fused_eh_norm 之后、eh_proj 之前对 eh_input 做 sp_shard（同步维护 padding 掩码），draft 输出经 shared_head.norm 后用 sp_all_gather 只恢复一次全量序列；非 SP 分支保留原有 tensor_model_parallel_all_reduce。
5. 测试配套：新增 tests/models/deepseek_v32/test_sequence_parallel.py，用 _IdentityNorm / _RecordingModule / _RecordingProjection 桩件和 _mock_sequence_parallel_collectives 模拟集合通信，验证两条数据契约：decoder 层 dense 状态下 attention 收到全量 3 token、MLP 只收到 shard 2 token 且两次连续调用保持一致；MTP 层 eh_proj 输入为 shard 2 token、最终输出恢复全量 3 token。

关键文件：

- vllm/models/deepseek_v32/nvidia/model.py（模块 模型层；类别 source；类型 core-logic；符号 DeepseekV32DecoderLayer.forward, DeepseekV32DecoderLayer.init, DeepseekV32Model.forward, _all_gather_sp_states）：核心实现：整层 use_sequence_parallel 取代仅 MoE 的开关，dense MLP 复制化，attention 前后改用 sp_all_gather/sp_reduce_scatter，删除逐层 gather 辅助函数，模型入口一次性 shard、出口打包 gather。
- vllm/models/deepseek_v32/nvidia/mtp.py（模块 推测解码；类别 source；类型 data-contract；符号 DeepseekV32MultiTokenPredictorLayer.forward）：MTP 推测解码链路：eh_proj 输入 shard 前置、padding 掩码同步维护、输出端一次性恢复全量序列，直接影响 speculative decoding 的 draft 生成数据流。
- tests/models/deepseek_v32/test_sequence_parallel.py（模块 回归测试；类别 test；类型 test-coverage；符号 test_decoder_layer_keeps_dense_states_sequence_sharded, test_mtp_projects_sequence_shard_and_restores_full_output, _mock_sequence_parallel_collectives）：新增回归测试，通过桩件与模拟集合通信锁定两条关键数据契约：decoder 层 dense 状态保持 shard（attention 全量、MLP shard），MTP 层 shard 输入并恢复 full 输出。

关键符号：DeepseekV32DecoderLayer.forward, DeepseekV32DecoderLayer.init, DeepseekV32Model.forward, DeepseekV32MultiTokenPredictorLayer.forward, sp_shard, sp_all_gather, sp_reduce_scatter, _all_gather_sp_states

关键源码片段

vllm/models/deepseek_v32/nvidia/model.py

核心实现：整层 use_sequence_parallel 取代仅 MoE 的开关，dense MLP 复制化，attention 前后改用 sp_all_gather/sp_reduce_scatter，删除逐层 gather 辅助函数，模型入口一次性 shard、出口打包 gather。

```
def forward(
    self,
    positions: torch.Tensor,
    hidden_states: torch.Tensor,
    residual: torch.Tensor | None,
) -> tuple[torch.Tensor, torch.Tensor]:
    # 序列并行开关从 " 仅 MoE 层 " 提升为整层：dense MLP 在 SP 下也保持
    # token shard 本地化，消除 dense/MoE 边界的 full/shard 状态转换。
    full_num_tokens = positions.shape[0]

    if residual is None:
        # 首层：hidden_states 为已 reduce 的 embedding，residual 从零建立。
        residual = hidden_states
        hidden_states = self.input_layernorm(hidden_states)
    elif self.use_sequence_parallel:
        # SP 模式：residual 与 hidden_states 都是本地 token shard，
        # 直接在 shard 上做 RMSNorm，跳过融合 all-reduce。
        hidden_states, residual = self.input_layernorm(hidden_states, residual)
    else:
        # 非 SP：上一层输出未 reduce，把 all-reduce 融合进本层
        # input_layernorm，避免一次独立的全量通信。
        hidden_states, residual = fused_allreduce_rms_norm(
            hidden_states, residual, self.input_layernorm
        )
    if self.use_sequence_parallel:
        # attention 需要完整序列视图，gather 全量 token；
        # sp_all_gather 内部处理 padding，再截断到真实长度。
        hidden_states = sp_all_gather(hidden_states)[:full_num_tokens]

    # self_attn 的 o_proj 以 reduce_results=False 运行，reduce 延后到
    # post_attention_layernorm 之前统一处理。
    hidden_states = self.self_attn(positions=positions, hidden_states=hidden_states)
    if self.use_sequence_parallel:
        # attention 输出 reduce 后重新切回本地 token shard；
        # 替代原先手工 pad + tensor_model_parallel_reduce_scatter。
        hidden_states = sp_reduce_scatter(hidden_states)
        hidden_states, residual = self.post_attention_layernorm(
            hidden_states, residual
        )
    else:
        hidden_states, residual = fused_allreduce_rms_norm(
            hidden_states, residual, self.post_attention_layernorm
        )

    # dense MLP 在 SP 下复制到每个 rank（构造时 is_sequence_parallel=True），
    # 与 Kimi K3、DeepSeek V4 数据流对齐；MoE 则按已分片输入执行。
    if self.use_sequence_parallel and isinstance(self.mlp, DeepseekV2MoE):
        hidden_states = self.mlp(hidden_states, already_sequence_parallel=True)
    else:
```

```
            hidden_states = self.mlp(hidden_states)
        return hidden_states, residual
```

vllm/models/deepseek_v32/nvidia/mtp.py

MTP 推测解码链路：eh_proj 输入 shard 前置、padding 掩码同步维护、输出端一次性恢复全量序列，直接影响 speculative decoding 的 draft 生成数据流。

```
# MTP 层：shard 发生在 eh_proj 之前，全量序列只在输出端恢复一次。
is_sequence_parallel = self.mtp_block.use_sequence_parallel
if is_sequence_parallel:
    # 同步维护 VLLM_MOE_SKIP_PADDING 的 padding 掩码，供稀疏
    # MoE（DSA）跳过 padding token 的计算。
    if envs.VLLM_MOE_SKIP_PADDING and is_forward_context_available():
        forward_context = get_forward_context()
        forward_context.is_padding = sp_padding_mask(
            forward_context.is_padding, eh_input
        )
    eh_input = sp_shard(eh_input)
hidden_states = run_glm52_plan(self._eh_plan, eh_input, self.eh_proj.weight)
if hidden_states is None:
    hidden_states = self.eh_proj(eh_input)
hidden_states, residual = self.mtp_block(
    positions=positions, hidden_states=hidden_states, residual=None
)
if not is_sequence_parallel:
    # 非 SP：MoE 输出未 reduce，这里显式 all-reduce。
    hidden_states = tensor_model_parallel_all_reduce(hidden_states)
# 复用 POST-final-norm hidden 作为下一个 draft step 的 previous_hidden_states，
# pre-norm 复用会拉低 MTP 接受率（与 deepseek_mtp.py 的 PR #45895 对齐）。
hidden_states, _ = self.shared_head.norm(hidden_states, residual)
if is_sequence_parallel:
    # 整个 MTP 链只在末尾恢复一次全量序列。
    hidden_states = sp_all_gather(hidden_states)[: positions.shape[0]]
return hidden_states, hidden_states
```

评论区精华

本 PR 没有实质的 review 讨论线程（review_comments_count = 0）。仅有的互动是：WoosukKwon 评论 /ci run 触发 Buildkite CI #82937，以及 claude[bot] 的自动提示评论（告知该仓库配置为手动 review，可留言 @claude review）。变更由作者直接合并，外部 reviewer 未提出质疑或设计取舍。

- 无实质 review 讨论 (other): 变更由作者直接合并，未发现外部 reviewer 的质疑或设计取舍讨论。

风险与影响

- 风险：

1. 显存占用增加：dense MLP 在 SP 下每 rank 复制，GLM-5.2 NVFP4 checkpoint 加载显存从 117.38 GiB 升至 118.01 GiB（+0.63 GiB），对显存紧张的部署是明确代价。
2. 核心执行路径变更：DeepseekV32DecoderLayer.forward 与 DeepseekV32Model.forward 是每 token 必经路径，集合通信从原始 TP collective 切到 sp_* 优化原语，首次执行有一次 custom-collective JIT 编译开销（benchmark 已排除该一次性成本）。
3. SP 与 PP 组合限制：新增 assert not self.use_sequence_parallel（PP 非末 rank 路径）与入口 assert residual is None，SP 与 PP 同时开启会直接断言失败；原实现同样要求 pipeline_parallel_size == 1，因此 PP 下行为不变，但需要确认该限制的文档化。
4. MTP 推测解码路径变更：eh_proj 输入 shard 化影响 draft 模型数值流，单测只验证 token 数契约，GSM8K 评估与端到端 benchmark 覆盖了 GLM-5.2 NVFP4，但未覆盖其他 DSv3.2 变体与更多 MTP 步数组合。
5. padding 掩码依赖：VLLM_MOE_SKIP_PADDING 环境变量未开启时 sp_padding_mask 路径不生效，同一代码在两种配置下的行为一致性需测试覆盖。 - 影响：对用户与系统：DeepSeek V3.2 / GLM-5.2 系列在 DP=2、TP=2、EP 场景下端到端吞吐提升 3.61%~5.22%，TPOT 降低 4.59%~5.06%，TTFT 略降，PIECEWISE CUDA graph 内存减少近半（1.21 → 0.67 GiB）；代价是每 worker 加载显存 +0.63 GiB。对团队：确立了 dsv32 全链路 SP 数据流与 sp_* 原语使用范式，为后续 DeepSeek V4、Kimi K3 等模型的 SP 统一实现提供参照；新增回归测试锁定了 dense 层保持 shard 与 MTP shard 契约，降低后续重构风险。影响面限于 nvidia 后端模型实现，rocm 等其余后端不受影响。 - 风险标记：核心执行路径变更，显存占用增加，SP 与 PP 不兼容，MTP 推测解码路径变更，集合通信 JIT 首启开销

关联脉络

- PR #51425 [Perf] Narrow DeepSeek V3.2 eager CUDA graph region: 同一 DeepSeek V3.2 模型性能优化线，改动 attention 与 nvidia/rocm 后端，与本 PR 共同降低 DSv3.2 执行延迟。
- PR #51298 [DSv32/GLM Perf] Skip short prefill topk for dense mha layer, 97.9% kernel level latency reduction: 同一模型（deepseek_v32）性能优化线，涉及 dense MHA 层与 MTP topk 路径，与本 PR 的 dense 层与 MTP 改动相邻。
- PR #50585 [K3 Perf] Optimize k3 dspark fused kv, 4.5~4.6x kernel performance improvement: Kimi K3 性能优化，本 PR 的 dense MLP SP 复制数据流以 Kimi K3 和 DeepSeek V4 为参照范式。
- PR #49793 MTP draft-decode path optimization (mentioned in PR body): PR body 明确排查过的外部 PR：优化通用 MTP draft-decode 路径，不改变 DeepSeek V3.2 序列并行状态流，与本 PR 不重叠。