PR #33480 完整报告

sgl-project/sglang

[AMD] Support prefill context parallel two batch overlap for DeepSeek V4

合并时间：2026-08-17 13:40

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33480

执行摘要

- 一句话：DeepSeek V4 新增 AMD prefill CP 与 TBO 重叠支持
- 推荐动作：值得精读，尤其是 split-phase async all-gather 与 duplicate communicator 避免 RCCL 死锁的设计。cp_utils.py 的 launch/finish 句柄（keepalive + event）和 compressor.py 的 prelaunch_kv_score 是可在其他模型上复用的异步通信模式。建议结合 perf_sweep_report §4.6 的结论，明确该特性对输入长度的收益阈值。

功能与动机

PR body 指出：DeepSeek V4 prefill CP 将长上下文 prefill 分布到 attention CP ranks，但引入了 per-layer CP collectives；此前 --enable-prefill-cp 与 --enable-two-batch-overlap 在未启用 DP attention 时会被拒绝，CP batch 也被运行时 TBO gate 显式排除。目标是在 AMD/ROCm 路径上把这些 CP 通信与另一子批次的 attention/MoE 计算重叠，以降低 TTFT 并提升长上下文 prefill 吞吐。

实现拆解

1. 并行组初始化扩展：parallel_state.py 新增 _ATTN_CP_OVERLAP 全局变量、get_attn_cp_overlap_group() 与 _init_attn_cp_overlap_group()，initialize_model_parallel 增加 duplicate_attn_cp_group 参数；bootstrap.py 在 is_hip 且同时开启 TBO 与 prefill CP 时传入 True。dp_attention.py 增加两个封装函数 attn_cp_overlap_all_gather_into_tensor / attn_cp_overlap_reduce_scatter_tensor，所有异步 CP 通信都走这个重复通信器。
2. 拆分式异步 CP 工具：cp_utils.py 新增 cp_all_gather_rerange_launch / cp_all_gather_rerange_finish。launch 在通信流上用 attn_cp_overlap 通信器发起 all-gather，用 _tbo_event 记录完成事件，返回 (output, input, event, cp_size) 句柄；finish 在消费前 wait_event 并做 round-robin 重排。句柄中的输入引用用于 keepalive，避免分配器在通信内核读取前复用输入内存。
3. DeepSeek V4 前向与门控：deepseek_v4.py 新增 _cp_tbo_launch 统一管理通信流、持久 buffer 与事件，新增 op_cp_gather_a/b、op_cp_moe、op_cp_combine_a/b 六个 op 组成 CP 专用 TBO 序列；新增 _forward_layers_tbo_cp 负责按子批次重建 CP 元数据、执行重叠、合并输出。_can_run_tbo 将 CP 路径限定为 HIP + round-robin split + CP-v1 + 非 EP TP-MoE + 单 PP + 可拆分 batch，并保留原有非 CP 逻辑。operations_strategy.py 的 init_new_tbo 增加 use_cp 参数并生成新的 op 调度列表。

4. KV 与 kv_score 预取：compressor.py 新增 prelaunch_kv_score 与 _pending_key，在 HIP 且存在 _cp_prefetch_comm_stream 时先算 kv_score 并发起 CP all-gather，等投影与压缩计算完成后由 compute_kv_score 收集结果；deepseek_v4.py 的 _forward_prepare 同样对 KV 使用 cp_all_gather_rerange_launch，并在 indexer/compressor 之后 finish。
5. 参数与对齐配套：server_args.py 放开 CP+TBO 同时开启限制，仅 HIP + round-robin-split 例外；two_batch_overlap.py 在 HIP 下把 TBO padding 对齐到 TP 与 CP 对齐大小的最小公倍数，确保拆分后每个子批次可被 CP 均匀分片。
6. 测试：新增 test/registered/amd/test_deepseek_v4_pro_fp4_cp_tbo.py，注册 nightly-amd-8-gpu-mi35x-deepseek-v4-pro 套件，以 GSM8K（1319 并发）0.92 精度底线验证 CP+TBO 与 CP-only 数值等价，并覆盖启动、并发通信器稳定性与子批次 CP 元数据。

关键文件：

- python/sglang/srt/models/deepseek_v4.py（模块 模型层；类别 source；类型 data-contract；符号 _cp_tbo_launch, op_cp_gather_a, op_cp_gather_b, op_cp_moe）：核心实现文件：新增 CP 专用 TBO op 序列（op_cp_gather_a/b、op_cp_moe、op_cp_combine_a/b）、_cp_tbo_launch 通信 launch 工具、_forward_layers_tbo_cp 拆分 / 合并流程，以及 _can_run_tbo 的 HIP CP 门控。
- test/registered/amd/test_deepseek_v4_pro_fp4_cp_tbo.py（模块 AMD 测试；类别 test；类型 test-coverage；符号 TestDeepseekV4ProFp4CPInterleaveTbo, setUpClass, tearDownClass, test_a_gsm8k）：新增 AMD 8-GPU nightly 测试，覆盖 CP+TBO 启动、1319 并发 GSM8K 精度、子批次 CP 元数据与并发通信器稳定性。
- python/sglang/srt/distributed/parallel_state.py（模块 并行组；类别 source；类型 core-logic；符号 get_attn_cp_overlap_group, _init_attn_cp_overlap_group）：新增第二个 CP 通信器 attn_cp_overlap，解决同一 RCCL 通信器被两个流同时驱动导致死锁的问题。
- python/sglang/srt/layers/utils/cp_utils.py（模块 CP 工具；类别 source；类型 core-logic；符号 cp_all_gather_rerange_launch, cp_all_gather_rerange_finish）：提供 split-phase 的 CP all-gather：cp_all_gather_rerange_launch 在通信流上异步发起并保持输入存活，cp_all_gather_rerange_finish 消费前再等待并做 round-robin 重排。
- python/sglang/srt/layers/attention/dsv4/compressor.py（模块 压缩器；类别 source；类型 core-logic；符号 _pending_key, prelaunch_kv_score）：新增 prelaunch_kv_score，让 kv_score 的 CP all-gather 在投影前发起、在 compute_kv_score 中收集，用 indexer/compressor 计算掩盖通信。
- python/sglang/srt/batch_overlap/operations_strategy.py（模块 重叠调度；类别 source；类型 core-logic；符号 _compute_moe_deepseek_v4_prefill）：为 CP 路径生成新的 op 序列：op_cp_gather_a/b、op_cp_moe、op_cp_combine_a/b，并断言仅支持非 EP TP-MoE。
- python/sglang/srt/layers/dp_attention.py（模块 DP 通信；类别 source；类型 core-logic；符号 attn_cp_overlap_all_gather_into_tensor, attn_cp_overlap_reduce_scatter_tensor）：新增两个对 attn_cp_overlap 通信器的封装，供 cp_utils 的异步工具使用。

- python/sglang/srt/batch_overlap/two_batch_overlap.py（模块 批重叠；类别 source；类型 dependency-wiring）：HIP 下 TBO padding 对齐到 TP 与 CP 对齐大小的最小公倍数，确保拆分后每个子批次可被 CP 均分。
- python/sglang/srt/distributed/bootstrap.py（模块 并行初始化；类别 source；类型 core-logic）：在 HIP + TBO + prefill CP 条件下向 initialize_model_parallel 传递 duplicate_attn_cp_group=True。
- python/sglang/srt/server_args.py（模块 参数校验；类别 source；类型 core-logic）：放开 CP 与 TBO 同时开启的校验，仅针对 HIP + round-robin-split 允许非 DP-attention 的 CP TBO 组合。

关键符号：_cp_tbo_launch, op_cp_gather_a, op_cp_gather_b, op_cp_moe, op_cp_combine_a, op_cp_combine_b, _cp_children_splittable, _setup_child_cp_metadata, _forward_layers_tbo_cp, prelaunch_kv_score, compute_kv_score, cp_all_gather_rerange_launch, cp_all_gather_rerange_finish, _init_attn_cp_overlap_group, get_attn_cp_overlap_group, attn_cp_overlap_all_gather_into_tensor, attn_cp_overlap_reduce_scatter_tensor, _compute_moe_deepseek_v4_prefill

关键源码片段

python/sglang/srt/models/deepseek_v4.py

核心实现文件：新增 CP 专用 TBO op 序列（op_cp_gather_a/b、op_cp_moe、op_cp_combine_a/b）、_cp_tbo_launch 通信 launch 工具、_forward_layers_tbo_cp 拆分 / 合并流程，以及 _can_run_tbo 的 HIP CP 门控。

```
def _cp_tbo_launch(self, state, x, key, out_rows, collective):
    # 仅 HIP 启用：CP 维度的 TBO MoE 通信只走 AMD/ROCm 路径
    assert _is_hip, 'CP+TBO MoE overlap 是 HIP-only'

    x = x.contiguous()
    sub = state.tbo_subbatch_index # 当前子批次索引（0 或 1）

    # 复用 TBO 持久缓冲区，避免每次 launch 重新分配
    out = get_tbo_persistent_buffer(
        (key, sub), out_rows, x.shape[1], x.dtype, x.device
    )

    # 通信流先等待计算流上的写入完成，再执行 collective
    comm = get_dp_tbo_comm_stream()
    comm.wait_stream(torch.cuda.current_stream())
    with torch.cuda.stream(comm):
        collective(out, x)
        event = _tbo_event((key, sub))
        event.record(comm) # 记录完成事件，供计算流稍后等待

    # 返回输出、事件与输入引用；输入引用用于 keepalive，
    # 防止 allocator 在通信内核读取前复用该内存块
```

```
        return out, event, x

    def op_cp_gather_a(self, state):
        # 子批次 A：发起 CP all-gather，不等待，把输入交给通信流
        local = state.pop('hidden_states_mlp_input')
        out, event, keepalive = self._cp_tbo_launch(
            state, local, 'cpgh',
            local.shape[0] * get_parallel().attn_cp_size,
            attn_cp_overlap_all_gather_into_tensor,
        )
        state.global_hidden = out
        state.cp_gather_event = event
        state.cp_gather_keepalive = keepalive

    def op_cp_gather_b(self, state):
        # 子批次 B：真正消费 all-gather 结果前再等待完成
        torch.cuda.current_stream().wait_event(state.pop('cp_gather_event'))
        state.pop('cp_gather_keepalive')

    def op_cp_moe(self, state):
        fb = state.forward_batch
        global_ids = fb._cp_moe_input_ids # 全局 token 顺序下的 MoE 输入 id
        with get_forward().scoped(mlp_reduce_scatter=True):
            state.global_expert_out = self.mlp(
                state.pop('global_hidden'), fb,
                input_ids=global_ids,
                input_ids_global=global_ids,
            )

    def op_cp_combine_a(self, state):
        # 子批次 A：发起 reduce-scatter，把专家输出分散回各 CP rank
        global_out = state.pop('global_expert_out')
        out, event, keepalive = self._cp_tbo_launch(
            state, global_out, 'cplo',
            global_out.shape[0] // get_parallel().attn_cp_size,
            attn_cp_overlap_reduce_scatter_tensor,
        )
        state.local_out = out
        state.cp_combine_event = event
        state.cp_combine_keepalive = keepalive

    def op_cp_combine_b(self, state):
        # 子批次 B：消费 reduce-scatter 结果前等待完成
        torch.cuda.current_stream().wait_event(state.pop('cp_combine_event'))
        state.pop('cp_combine_keepalive')
        state.hidden_states_mlp_output = state.pop('local_out')
```

python/sglang/srt/layers/utils/cp_utils.py

提供 split-phase 的 CP all-gather：cp_all_gather_rerange_launch 在通信流上异步发起并保持输入存活，cp_all_gather_rerange_finish 消费前再等待并做 round-robin 重排。

```
def cp_all_gather_rerange_launch(input_tensor, cp_size, comm_stream, event_key):
    # 在 comm_stream 上发起 round-robin CP all-gather，不等待完成
    # 与 cp_all_gather_rerange_finish 配对使用。把 launch 与 wait 拆开，
    # 是让 attention 侧 CP gather 得以与其它计算重叠的唯一方式：
    # 如果发出与等待在同一处执行，只是移动队列位置，并不会产生重叠。
    from sglang.srt.distributed.parallel_state import (
        get_attn_cp_group,
        get_attn_cp_overlap_group,
    )

    group = get_attn_cp_overlap_group()
    assert group is not get_attn_cp_group(), (
        '通信流路径必须使用独立的 attn_cp_overlap 通信器；'
        '从两个流驱动同一个 RCCL 通信器会死锁'
    )

    input_tensor = input_tensor.contiguous()
    with use_symmetric_memory(group, disabled=not is_allocation_symmetric()):
        output_tensor = input_tensor.new_empty(
            (input_tensor.shape[0] * cp_size, *input_tensor.shape[1:]),
        )

    # 通信流等待当前计算流上的写入完成，再读取输入
    comm_stream.wait_stream(torch.cuda.current_stream())
    with torch.cuda.stream(comm_stream):
        attn_cp_overlap_all_gather_into_tensor(output_tensor, input_tensor)
        event = _tbo_event(event_key)
        event.record(comm_stream)

    # 返回（输出，输入引用，事件，cp_size）句柄；输入引用用于 keepalive
    return (output_tensor, input_tensor, event, cp_size)

def cp_all_gather_rerange_finish(handle):
    # 在当前流上等待已发起的 gather 完成，然后做 round-robin 重排
    output_tensor, _keepalive, event, cp_size = handle
    torch.cuda.current_stream().wait_event(event)
    out_shape = output_tensor.shape
    # 从 (cp_size, local_len, ...) 转成全局 token 顺序
    return (
        output_tensor.view(cp_size, -1, *out_shape[1:])
        .transpose(0, 1)
        .reshape(out_shape)
    )
```

python/sglang/srt/layers/attention/dsv4/compressor.py

新增 prelaunch_kv_score，让 kv_score 的 CP all-gather 在投影前发起、在 compute_kv_score 中收集，用 indexer/compressor 计算掩盖通信。

```
def _pending_key(self):
    # 以 (kv_score, layer_id, is_in_indexer) 作为 pending 字典的键，
    # 区分主压缩器与 indexer 内部的压缩器
    return ('kv_score', self.layer_id, self.is_in_indexer)

def prelaunch_kv_score(self, x, forward_batch):
    # 提前计算 kv_score 并立即发起 CP all-gather，不等待完成
    # kv_score 只依赖 x，而 x 在进入 attention 时就已经就绪，
    # 所以可以在 q/kv 投影之前发起 gather，由投影计算隐藏通信延迟；
    # 之后 compute_kv_score 里再收集结果。
    if not _is_hip:
        return # 该优化只在 AMD/ROCm 上启用

    # 只有 CP+TBO 路径会预置 _cp_prefetch_comm_stream
    comm_stream = getattr(forward_batch, '_cp_prefetch_comm_stream', None)
    if comm_stream is None or not dsa_use_prefill_cp(forward_batch):
        return

    kv_score = linear_bf16_fp32(x, self.wkv_gate.weight)

    # 以 forward_batch 为键挂到 pending 字典：每个 TBO 子批次持有独立的
    # forward_batch，因此两个子批次不会互相取到对方的 gather 结果
    pending = forward_batch.__dict__.setdefault('_cp_pending_gathers', {})
    pending[self._pending_key()] = cp_all_gather_rerange_launch(
        kv_score, get_parallel().attn_cp_size, comm_stream, self._pending_key()
    )

def compute_kv_score(self, x, forward_batch):
    # HIP 路径优先取回已提前发起的 gather 结果
    if _is_hip:
        pending = getattr(forward_batch, '_cp_pending_gathers', None)
        handle = pending.pop(self._pending_key(), None) if pending else None
        if handle is not None:
            return cp_all_gather_rerange_finish(handle)

    kv_score = linear_bf16_fp32(x, self.wkv_gate.weight)

    # CUDA 路径：未使用预取机制，仍然委托给 backend 同步执行
    if dsa_use_prefill_cp(forward_batch):
        kv_score = cp_materialize_global_token_order(
            kv_score,
            forward_batch,
            torch.cuda.current_stream(),
        )
    return kv_score
```

评论区精华

唯一 review 评论是 HaiShaw 的 APPROVED（LGTM），没有形成实质性代码讨论。Issue 评论中有三条信息：HaiShaw 建议 @1am9trash 考虑更新 cookbook 并请求 @Duyi-Wang 与 @At1a8 确认；michaelzhang-ai 报告 CI 中 Triton 3.4.0 AMD backend（gfx950）在 lowering FP8 extend-attn 时崩溃，影响 test_qwen3_5_fp8_kv_cache，并提示该测试由本 PR（#33480）添加，建议与 Triton >=3.6 做 A/B 验证；gemini-code-assist[bot] 仅发布停服通知，无实质审查内容。

- 更新 cookbook 文档（CP+TBO 用法） (documentation): 未看到后续变更，文档更新未在本 PR 内落地。
- Triton AMD PassManager 崩溃归因 (correctness): 未在本 PR 讨论中确认根因；需验证是否为环境 /Triton 版本问题。
- Gemini Code Assist 停服通知 (other): 无实质内容，不构成审查意见。

风险与影响

- 风险：死锁与资源：同一 RCCL 通信器被多个流并发驱动会死锁，本 PR 用 duplicate communicator 规避，但新增通信器会占用更多 RCCL 资源，8-GPU 场景下初始化与内存开销需要监控。性能边界：TBO 收益与输入长度强相关，8k/1k 时吞吐下降 3.6%、TTFT 上升 9.1%，64k 后才稳定获得 7%~12% 的 TTFT 收益，用户需按实际请求长度选择是否开启。精度与数值：GSM8K 从 CP-only 的 0.944 降到 CP+TBO 的 0.936，仍高于 0.92 底线，但 FP4 量化与异步重叠组合下的数值漂移需要持续观察。平台与后端限定：整个路径仅对 HIP + unified_kv_triton + DSA round-robin split 生效，前置条件不满足时回退 eager；若上层拆分逻辑改变，keepalive 依赖可能失效。CI 不确定性：Triton PassManager 崩溃的归因尚未确认，建议追查是否为 Triton 3.4.0 既有问题，避免对本 PR 结论产生误判。
- 影响：用户侧：AMD/ROCm 用户开启 --enable-prefill-cp 与 --enable-two-batch-overlap 后，长上下文（32k 以上）prefill 吞吐提升 3%~14%，TTFT 降低 7%~12%；8k 以下短输入可能回退。系统侧：新增第二个 CP 通信器与多流调度，影响 RCCL 资源与内存占用；通过 gate 保证 CUDA 与非 CP 路径行为不变。团队侧：需要维护 CP+TBO 与现有 TBO 双路径，新测试注册到 nightly-amd-8-gpu-mi35x 套件，CI 时长与维护成本增加。
- 风险标记：RCCL 双流死锁风险，HIP-only 专属路径，短输入性能回退 (8k)，GSM8K 精度下降 (0.936)，CI Triton 崩溃归因不明，多通信器资源开销

关联脉络

- PR #35382 [Refactor] Share the page-aligned decode alloc lens between EAGLE and DFLASH: 同属 token 对齐与分配水位逻辑；本 PR 在 two_batch_overlap.py 中引入 TP/CP 双对齐，与 EAGLE/DFLASH 页对齐分配水位属于同一类对齐机制演进。
- PR #35286 [Fix] Assert the page-aligned SWA evict floor at PD decode prealloc: PD 解码预分配页对齐断言属 AMD 平台内存对齐一致性工作，与本 PR 的 TBO padding 对齐目标一致。

- PR #35396 [Fix] Assert the page-aligned SWA evict floor on both PD decode prealloc paths: 在两条 PD prealloc 路径统一页对齐断言，与本 PR 的 CP/TBO 对齐逻辑互补，避免不同路径对齐口径不一致。
- PR #35339 [diffusion] Per-request lossy accelerations: Cache-DiT, CFG gating, attention backend override: per-request 有损加速开关重构与 TBO 同属性能优化开关体系；虽然领域在 diffusion，但展示了按请求控制加速路径的思路。