PR #23185 完整报告

sgl-project/sglang

[Bugfix] Fix DeepEP timeout when compiling DeepGeMM in EP+DP+TP

合并时间：2026-04-20 08:36

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23185

执行摘要

- 一句话：修复 DeepGeMM 编译阶段 DeepEP 超时问题，通过全局同步屏障和全 DP 预热请求确保跨节点编译一致性。
- 推荐动作：该 PR 值得精读，尤其是 _deepep_precompile_tp_barrier 的设计和屏障插入位置，展示了如何在第三方库（DeepEP）超时机制不足时，利用上层同步原语（torch.distributed）进行保护。同时，预热请求的改造体现了对 DP 并行度感知的编译触发策略。

功能与动机

PR body 明确指出，在 Kimi K2 EP+DP+TP 32 配置下编译 DeepGeMM 时遇到 DeepEP 超时错误。根本原因是编译仅发生在每个节点的 local_rank=0 上，且不同节点编译速度可能差异很大（例如大块预填充尺寸放大差异）。DeepEP 的 all-to-all 操作超时时间远短于 torch.distributed，当编译时间差超过 DeepEP 超时（如 100 秒）就会触发超时。

实现拆解

1. 在 DeepEP 调度核心路径添加编译阶段同步屏障(python/sglang/srt/layers/moe/token_dispatcher/deepep.py):
   - 新增 _deepep_precompile_tp_barrier 函数，检查环境变量 SGLANG_IN_DEEPGEMM_PRECOMPILE_STAGE，仅在编译阶段执行 get_tp_group().barrier()。
   - 在 _dispatch_core 和 _combine_core 函数中调用此屏障，确保 DeepEP 的 buffer.dispatch 和 buffer.combine 操作前所有 TP 组内 rank 已同步。
   - 导入 get_tp_group 以获取同步组。
2. 修改编译预热逻辑以覆盖所有 DP 排名(python/sglang/compile_deep_gemm.py):
   - 在 warm_up_compile 函数中，根据 server_args.dp_size 动态构造输入 ID 列表和 bootstrap 字段，为每个 DP 排名生成独立的请求数据。
   - 在 launch_server_process_and_send_one_request 函数中，类似地修改 rank-0 节点发送的同步请求，确保 payload 包含所有 DP 排名的数据。
   - 这样每个节点都能在自定义预热阶段触发编译，避免只有第一个节点有缓存导致的运行时编译时间巨大差异。

3. 无测试或配置配套改动：本次变更未包含直接对应的测试文件更新，也未涉及部署或配置文件的修改。

关键文件：

- python/sglang/srt/layers/moe/token_dispatcher/deepep.py（模块 MoE 调度器；类别 source；类型 core-logic；符号 _deepep_precompile_tp_barrier, DeepEPPDispatchHooks）：DeepEP 调度器的核心实现文件，直接修复超时问题的同步屏障在此添加。
- python/sglang/compile_deep_gemm.py（模块 编译工具；类别 source；类型 core-logic；符号 warm_up_compile, launch_server_process_and_send_one_request）：DeepGeMM 编译预热脚本，修改后确保所有 DP 排名都能在预热阶段触发编译。

关键符号：_deepep_precompile_tp_barrier, warm_up_compile, launch_server_process_and_send_one_request

关键源码片段

python/sglang/srt/layers/moe/token_dispatcher/deepep.py

DeepEP 调度器的核心实现文件，直接修复超时问题的同步屏障在此添加。

```
def _deepep_precompile_tp_barrier() -> None:
    # DeepEP 的 all-to-all 操作超时远短于 torch.distributed,
    # 如果不同 rank 编译速度不同，可能快速触发超时。
    # 为避免此问题，我们在编译阶段使用 torch.distributed 的屏障。
    # 仅在编译阶段应用此屏障，防止运行时额外的 all-reduce 开销。
    if envs.SGLANG_IN_DEEPGEMM_PRECOMPILE_STAGE.get():
        get_tp_group().barrier() # 使用 TP 组进行同步，确保组内 rank 在编译阶段对齐


class DeepEPPDispatchHooks(DispatcherBaseHooks):
    def __call__(self, dispatcher: BaseDispatcher):
        for hook_fun in self.hook_dict.values():
            hook_fun(dispatcher)


class DeepEPDispatcher(BaseDispatcher):
    # ... 其他方法 ...

    def _dispatch_core(
        self,
        x: torch.Tensor,
        topk_ids: torch.Tensor,
        topk_weights: torch.Tensor,
        previous_event,
    ):
        buffer = self._get_buffer()
        # 获取调度布局 ...
        # FIXME: `handle` 应随 token 从 dispatch 传输到 combine，但当前作为成员变量工作。
```

```
        _deepep_precompile_tp_barrier() # 关键：在调用 DeepEP buffer.dispatch 前插入同步屏障
        (
            recv_x,
            recv_topk_ids,
            recv_topk_weights,
            num_recv_tokens_per_expert,
            self.handle,
            event,
        ) = buffer.dispatch(
            x,
            topk_idx=topk_ids,
            topk_weights=topk_weights,
            num_tokens_per_rank=num_tokens_per_rank,
            num_tokens_per_rdma_rank=num_tokens_per_rdma_rank,
            is_token_in_rank=is_token_in_rank,
            num_tokens_per_expert=num_tokens_per_expert,
            previous_event=previous_event,
            async_finish=self.async_finish,
            allocate_on_comm_stream=(previous_event is not None) and self.async_finish,
            expert_alignment=128 if deep_gemm_wrapper.ENABLE_JIT_DEEPGEMM else 1,
            config=DeepEPConfig.get_instance().normal_dispatch_config,
        )
        # ... 返回结果 ...
```

python/sglang/compile_deep_gemm.py

DeepGeMM 编译预热脚本，修改后确保所有 DP 排名都能在预热阶段触发编译。

```
@warmup("compile-deep-gemm")
async def warm_up_compile(
    disaggregation_mode: str, tokenizer_manager: TokenizerManager
):
    print("\nGenerate warm up request for compiling DeepGEMM...\n")
    server_args = tokenizer_manager.server_args
    dp_size = server_args.dp_size # 获取数据并行大小
    base_ids = [0, 1, 2, 3]
    sampling_params = {
        "temperature": 0.0,
        "max_new_tokens": 8,
        "ignore_eos": True,
    }

    if disaggregation_mode != "null":
        # 在 disaggregation 模式下，为每个 DP 排名构造独立的输入 ID 列表和 bootstrap 字段
        input_ids = [list(base_ids) for _ in range(dp_size)]
        generate_req_input = GenerateReqInput(
            input_ids=input_ids,
            sampling_params=sampling_params,
        )
        generate_req_input.bootstrap_host = [FAKE_BOOTSTRAP_HOST] * dp_size
```

```
            generate_req_input.bootstrap_room = [
                i * (2**63 // dp_size) + (i % server_args.tp_size) for i in range(dp_size)
            ]
        else:
            # 非 disaggregation 模式，根据 DP 大小调整输入格式
            input_ids = (
                base_ids if dp_size == 1 else [list(base_ids) for _ in range(dp_size)]
            )
            generate_req_input = GenerateReqInput(
                input_ids=input_ids,
                sampling_params=sampling_params,
            )

        await tokenizer_manager.generate_request(generate_req_input, None).__anext__()
```

评论区精华

review 中仅有一条来自 gemini-code-assist[bot] 的评论，指出使用 get_tp_group() 进行屏障同步在 TP 为节点内配置时可能不足，建议改用 get_world_group() 以确保跨节点全局同步。但最终 PR 被合并时未采纳此建议，保持了 get_tp_group() 的实现。这可能意味着当前场景下 TP 组内同步已足够，或存在其他约束。

- 同步屏障使用 TP 组是否足够 (correctness): PR 最终未采纳建议，保持了 get_tp_group() 的实现，可能当前场景下 TP 组内同步已满足需求。

风险与影响

- 风险：
  1. 性能风险：新增的屏障仅在编译阶段激活，通过环境变量控制，避免了运行时额外开销，风险较低。
  2. 正确性风险：如果 TP 组未覆盖所有需要同步的 rank（例如跨节点场景），屏障可能无法完全消除编译时间差，导致超时问题残留。
  3. 兼容性风险：修改了预热请求的数据结构，需确保所有调用方和下游处理能正确解析多 DP 排名的输入格式。
  4. 回归风险：对 DeepEP 核心调度路径的修改（添加屏障调用）需仔细验证，避免引入死锁或同步错误。
- 影响：
  1. 对用户影响：解决了特定并行配置下编译失败的问题，提升了 DeepGeMM 在大型集群上的编译成功率和稳定性，用户无需手动调整超时或重试。
  2. 对系统影响：编译阶段增加同步屏障可能轻微延长编译总时间，但避免了超时导致的编译失败和重试，整体上提高了系统可靠性。
  3. 对团队影响：提供了处理分布式编译同步问题的模式，未来类似场景可参考此方案。 - 风险标记：核心路径变更，分布式同步风险，缺少测试覆盖

关联脉络

- PR #22850 [AMD] Reduce NSA indexer kernels (weights_proj, k-cache store kernel fusion): 同属性能优化和调度相关，涉及 MoE 和内核融合，但本 PR 聚焦编译阶段超时修复。
- PR #23019 refactor(moe): de-duplicate triton MoE runner path into shared helpers: 同属 MoE 模块重构，但本 PR 是 bugfix 而非重构。