PR #12961 完整报告

sgl-project/sglang

Fix DP attention on CPU

合并时间：2026-08-19 09:56

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/12961

执行摘要

- 一句话：CPU 引擎 DP 注意力功能修复，含 MoE 归约与拷贝路径
- 推荐动作：值得精读。该 PR 展示了“先参考实现保功能、再提交优化 kernel”的跨平台适配策略，且 review 中关于循环导入、hasattr 误判、全局函数分发的讨论有实际工程借鉴价值。对 CPU 平台维护者尤其有参考意义，建议关注后续 sgl-kernel 对 memcpy_cpu、reduce_scatter_tensor、all_gather_into_tensor 的替换。

功能与动机

DP attention 此前只在 CUDA 路径可用，CPU 引擎开启 --enable-dp-attention 时存在多个断点：memcpy_triton 依赖 Triton 内核无法在 CPU 上执行，parallel_state 的 custom op 只对 CUDA/XPU 注册，DeepSeek MoE 的 all_reduce 判断未考虑 use_reduce_scatter 导致结果错误。PR body 明确表示“Fix the functionality of DP attention on CPU”，并说明优化 kernel 留待后续提交。

实现拆解

实现按“先保功能、后做优化”的策略拆解为五步：

1. 通信原语 CPU 化：在 python/sglang/srt/distributed/parallel_state.py 的 reduce_scatter_tensor 与 all_gather_into_tensor 中，将条件从 _is_npu 扩展为 _is_npu or _is_cpu，让 CPU 走 _reduce_scatter_tensor / _all_gather_into_tensor 内部 torch.distributed.reduce_scatter_tensor / all_gather_into_tensor 的 torch 原生路径，绕过仅为 CUDA/XPU 注册的 custom op（reg_reduce_scatter_tensor / reg_all_gather_into_tensor）。这保证功能正确，但性能优化留待后续 kernel。
2. 数据搬运 CPU 化：在 python/sglang/srt/layers/dp_attention.py 中新增 memcpy_cpu 参考实现（基于 Tensor.copy_ 的按行段拷贝），并通过模块级 memcpy_func = memcpy_cpu if _is_cpu else memcpy_triton 做一次性分发，封装统一入口 memcpy，替换 _dp_gather_via_all_reduce 与 dp_scatter 中对 memcpy_triton 的直接调用。
3. attention TP 尺寸修正：在 python/sglang/srt/layers/attention/intel_amx_backend.py 中，num_head 计算从 model_runner.ps.tp_size 改为 get_parallel().attn_tp_size。启用 DP attention 时 attn_tp_size = tp_size // dp_size，原先的写法会导致注意力头数切分错误。

4. forward batch 的 CPU 适配：在 python/sglang/srt/model_executor/forward_batch_info.py 中，pin_memory 与 non_blocking 仅在非 CPU 时启用（self.use_pin_memory = not _is_cpu），避免 CPU 上不支持的 pin 内存路径；seq_len_fill_value 按 _is_cpu 选择 get_cpu_graph_seq_len_fill_value / get_cuda_graph_seq_len_fill_value，避免引入循环导入与 hybrid 注意力后端的 hasattr 误判。
5. MoE 归约对齐 GPU：在 python/sglang/srt/models/deepseek_v2.py 的 forward_cpu 中，将裸条件 not get_forward().fuse_mlp_allreduce 替换为 not should_skip_post_experts_all_reduce(is_tp_path=True)，与 CUDA 路径一致：当 DP + use_reduce_scatter 开启时，MoE kernel 输出已做 reduce-scatter，不再做冗余 all_reduce。

测试配套：在 test/registered/cpu/test_intel_amx_attention_backend_a.py 中新增 TestDPAttention 类，以 --tp 2 --enable-dp-attention --dp 2 启动服务并跑 GSM8K 评估，断言 accuracy 大于 0.7，注册到 CPU CI suite（register_cpu_ci）。

关键文件：

- python/sglang/srt/layers/dp_attention.py（模块 DP 注意力；类别 source；类型 core-logic；符号 memcpy_cpu, memcpy）：DP attention 的核心数据搬运逻辑：新增 CPU 参考实现 memcpy_cpu，并将 memcpy_triton 调用统一收敛到 memcpy 分发入口，是本次功能修复的主战场。
- test/registered/cpu/test_intel_amx_attention_backend_a.py（模块 CPU 测试；类别 test；类型 test-coverage；符号 TestDPAttention, setUpClass, tearDownClass, test_dp_attention_DP2TP2）：新增 TestDPAttention 端到端测试，以 TP=2 DP=2 启动 intel_amx 后端服务并跑 GSM8K，是本次 CPU DP 功能修复的验收依据。
- python/sglang/srt/model_executor/forward_batch_info.py（模块 批信息；类别 source；类型 data-contract）：修正 CPU 上 forward batch 的 pin_memory 与 seq_len_fill_value 选择逻辑，改动虽小但涉及数据契约，且 review 中循环导入与 hasattr 陷阱集中在此文件。
- python/sglang/srt/models/deepseek_v2.py（模块 DeepSeek 模型；类别 source；类型 data-contract）：修复 CPU 上 DeepSeek MoE 在 DP 开启时的冗余 all_reduce 问题，与 GPU 路径的 should_skip_post_experts_all_reduce 语义对齐，直接影响推理结果的正确性。
- python/sglang/srt/distributed/parallel_state.py（模块 并行状态；类别 source；类型 core-logic）：分布式集合通信的 CPU 分支：reduce_scatter_tensor 与 all_gather_into_tensor 在 CPU 上改走 torch.distributed 原生实现，是 DP 通信链路可用的基础。
- python/sglang/srt/layers/attention/intel_amx_backend.py（模块 注意力后端；类别 source；类型 dependency-wiring）：修复 IntelAMX 后端 num_head 计算使用的 TP 尺寸，DP 开启时需用 attn_tp_size（tp_size // dp_size），否则注意力头切分错误。

关键符号：memcpy_cpu, memcpy, reduce_scatter_tensor, all_gather_into_tensor, forward_cpu, test_dp_attention_DP2TP2

关键源码片段

python/sglang/srt/layers/dp_attention.py

DP attention 的核心数据搬运逻辑：新增 CPU 参考实现 memcpy_cpu，并将 memcpy_triton 调用统一收敛到 memcpy 分发入口，是本次功能修复的主战场。

```
# 文件：python/sglang/srt/layers/dp_attention.py
# CPU 上 Triton 内核不可用，提供基于 torch.copy_ 的参考实现；
# 后续计划用 sgl-kernel（AVX512）替换，见 TODO 注释。

# TODO: write c++ kernel for cpu
def memcpy_cpu(dst, src, dim, offset, sz, offset_src):
    # dp_attention 的 gather/scatter 只需要在 batch 维度拷贝一段连续行
    assert dim == 0, "Only dim=0 supported"
    assert src.shape[1:] == dst.shape[1:], "src and dst must have same trailing shape"

    total_rows_dst, total_rows_src = dst.shape[0], src.shape[0]
    dst_start, src_start = 0, 0

    # offset_src=True 表示从 src[offset:] 拷到 dst[0:]（scatter 方向）；
    # False 表示从 src[0:] 拷到 dst[offset:]（gather 方向）。
    if offset_src:
        src_start = offset
        dst_start = 0
    else:
        src_start = 0
        dst_start = offset

    # 裁剪边界，避免越界或读到 padding 外的数据
    dst_end = min(dst_start + sz, total_rows_dst)
    src_end = min(src_start + sz, total_rows_src)
    actual_sz = min(dst_end - dst_start, src_end - src_start)

    if actual_sz <= 0:
        return

    dst[dst_start : dst_start + actual_sz].copy_(
        src[src_start : src_start + actual_sz]
    )


# 全局一次决策，避免每次调用都走 if-else（review 中 mingfeima 的建议）
_is_cpu = is_cpu()
memcpy_func = memcpy_cpu if _is_cpu else memcpy_triton


def memcpy(dst, src, dim, offset, sz, offset_src):
    # 统一入口：GPU 走 Triton 内核，CPU 走参考实现
    memcpy_func(dst, src, dim, offset, sz, offset_src)
```

python/sglang/srt/distributed/parallel_state.py

分布式集合通信的 CPU 分支：reduce_scatter_tensor 与 all_gather_into_tensor 在 CPU 上改走 torch.distributed 原生实现，是 DP 通信链路可用的基础。

```
# 文件：python/sglang/srt/distributed/parallel_state.py
# NPU 与 CPU 都走 torch.distributed 后端（CPU 无 pynccl、无 custom op）；
# CUDA/XPU 才走注册的 custom op，保证对 Dynamo 图捕获不透明。

def reduce_scatter_tensor(self, output, input):
    if _is_npu or _is_cpu:
        # TODO: add optimized reduce_scatter_tensor kernel for cpu
        self._reduce_scatter_tensor(output, input)
    elif self._maybe_aiter_reduce_scatter(output, input):
        return
    else:
        reg_reduce_scatter_tensor(output, input, group_name=self.unique_name)


def all_gather_into_tensor(self, output, input):
    if _is_npu or _is_cpu:
        # TODO: add optimized all_gather_into_tensor kernel for cpu
        self._all_gather_into_tensor(output, input)
    else:
        # XPU 和 CUDA 都通过 custom op 保持对 Dynamo 不透明，
        # 直接调用 torch.distributed 会被改写并触发 sycl_event.wait() 破坏图捕获。
        reg_all_gather_into_tensor(output, input, group_name=self.unique_name)
```

评论区精华

Review 由 Intel 侧维护者 mingfeima 主导，经历两轮 CHANGES_REQUESTED 后 APPROVED，整体评价“generally LGTM, just some minor changes needed”，并建议“let's provide sgl-kernels as well and evaluate the performance internally first”。核心讨论点如下：

- attn_tp_size 语义辨析：mingfeima 询问 get_attention_tp_size() 与 model_runner.tp_size 的区别，chunyuan-w 解释 DP attention 关闭时 attn_tp_size = tp_size，开启时 attn_tp_size = tp_size // dp_size，且 triton_backend 等其它后端也统一使用该函数，结论是改用 get_parallel().attn_tp_size。
- memcpy 分发方式：mingfeima 连续两次建议把函数选择提为全局变量以省掉每次调用的 Python if-else，chunyuan-w 采纳并改为模块级 memcpy_func。
- 循环导入与 hasattr 陷阱：mingfeima 最初建议在 forward_batch_info 中用 hasattr(attn_backend, "get_cpu_graph_seq_len_fill_value") 判断，但 chunyuan-w 实测发现 HybridLinearAttnBackend 无条件同时定义 CPU 与 CUDA 两个方法，GPU 上会误调 CPU 方法，并且直接 import IntelAMXAttnBackend 会造成循环导入，最终改用模块级 _is_cpu 判断，这一取舍是 review 中最有价值的讨论。
- MoE 归约复用：mingfeima 指出应直接复用 GPU 路径上的 should_skip_post_experts_all_reduce，chunyuan-w 照做。

- 性能展望：mingfeima 评论 memcpy_cpu“用 AVX512 很容易实现，后续用 sgl-kernel 替换”，代码保留 TODO。
  - intel_amx_backend 中 tp_size 与 attn_tp_size 的语义辨析 (question): 改用 get_parallel().attn_tp_size，与其它注意力后端统一。
  - memcpy 分发的运行时开销与全局化建议 (design): 采用模块级 memcpy_func = memcpy_cpu if _is_cpu else memcpy_triton 一次性决策。
  - forward_batch_info 循环导入与 hybrid 后端的 hasattr 误判 (correctness): 改用模块级 _is_cpu 决定调用 get_cpu_graph_seq_len_fill_value 还是 get_cuda_graph_seq_len_fill_value，既避免循环导入也避免 hybrid 后端误判。
  - deepseek_v2 复用 should_skip_post_experts_all_reduce (design): 使用 should_skip_post_experts_all_reduce(is_tp_path=True) 统一 DP/ 非 DP 的归约判断。
  - memcpy_cpu 后续 AVX512/sgl-kernel 优化方向 (performance): 保留 TODO 注释，功能先行，优化 kernel 留待 follow-up PR。
  - pin_memory 的 if-else 精简 (style): 按评论建议精简为单表达式赋值。

风险与影响

- 风险：
  1. 性能风险：memcpy_cpu 是纯 Python + Tensor.copy_ 的逐段拷贝，无并行优化；在 decode 阶段 DP 通信频繁时可能成为 CPU 推理瓶颈。PR 自述与 review 均已确认后续会提交 AVX512/sgl-kernel 优化。
  2. 集合通信无优化 kernel：parallel_state.py 的 CPU 分支直接走 torch.distributed 原生实现，功能正确但无 custom kernel 加速；对 CPU 引擎影响可控，但不排除大 TP/DP 组合下通信占比过高。
  3. deepseek_v2.forward_cpu 影响面：变更影响所有 CPU 上运行的 DeepSeek MoE 模型，不限于 DP 场景。若 should_skip_post_experts_all_reduce 在非 DP 配置下返回值与原先条件不一致，可能导致回归，需要确认其默认行为（GPU 路径已长期验证，风险较低）。
  4. 测试覆盖偏窄：UT 仅覆盖 DeepSeek-Coder-V2-Lite-Instruct + intel_amx backend + DP2TP2 + GSM8K，未覆盖 MLA、更大并行度组合、以及非 AMX 的 CPU 注意力后端。 - 影响：对用户而言，CPU 平台首次打通了 --enable-dp-attention 的完整推理链路，DeepSeek 系 MoE 模型可以在 CPU 集群上享受 DP 带来的吞吐扩展。对系统而言，分布式通信层（parallel_state.py）和 forward batch 填充逻辑新增 CPU 分支，但不影响 GPU/NPU 既有路径。对团队而言，这是 Intel CPU 团队持续对齐 GPU 能力的一环，后续大概率有优化 kernel 的 follow-up PR；review 中已约定补充 cookbook 文档。 - 风险标记：CPU 集合通信与 memcpy 无优化 kernel，deepseek_v2 forward_cpu 条件变更影响非 DP 场景，测试覆盖仅 DP2TP2 单模型单后端，pin_memory 行为在 CPU 上被禁用需留意

关联脉络

- PR #34862 [Doc] Fix TP and attention-TP group layout in initialize_model_parallel docstring: 同为 attention-TP 与全量 TP 布局语义相关，本 PR 的 attn_tp_size 修正与之呼应，可对照理解 DP 下 attention 并行组的划分规则。
- PR #35061 [Fix] Select custom all-reduce v2 by topology capability: 同为分布式集合通信层（parallel_state.py 相关）的修复，说明不同硬件平台对集合通信实现的选择是持续演进的主题。
- PR #34923 Apply latest DeepEP branch: 涉及 DeepSeek MoE 的通信与依赖升级，与本 PR 的 deepseek_v2 归约路径修复同属 DeepSeek MoE 在分布式下的正确性保障。