PR #30915 完整报告

sgl-project/sglang

[Feature] Megatron LayerNorm sequence parallelism (--enable-layernorm-sp)

合并时间：2026-09-01 10:27

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30915

执行摘要

- 一句话：LayerNorm 序列并行：prefill 激活内存与延迟双降
- 推荐动作：值得精读，尤其三个设计决策：(1) CUDA graph 下捕获区内写入的 Python 标志在回放时过期这一问题的识别与 runs_sp 规避方案；(2) LayerCommunicator 构造全 SCATTERED sibling 的委托模式，避免在每个模型里打点；(3) fused matmul+collective 的可用性探测与普通 collective fallback 策略，兼顾性能与跨 torch 版本兼容性。对后续做任何序列切分、通信融合类特性都有直接参考价值。

功能与动机

长上下文 prefill 的激活显存与延迟随序列长度线性增长。PR body 指出，纯 TP 下 row-parallel 的 all_reduce 代数上等价于 reduce_scatter + all_gather，两种形式移动相同字节数——拆开后 LayerNorm/residual 只需在 1/tp 的 token 上计算，既降低激活内存又无额外通信开销。设计目标明确为 off by default and byte-identical when off，避免影响现有用户；实测 2×H100 上 GSM8K 0.904 vs 0.906（fp 重排噪声），prefill 0.28173s → 0.27180s（-3.5%）；2×B200 上 0.14388s → 0.13589s（-5.6%）；早期 8×H100（Qwen3-32B、32K）版本为 -16.5% 延迟、-27.7% 激活内存。

实现拆解

1. 新增 SP 核心模块 python/sglang/srt/layers/layernorm_sp.py（+245 行）：集中承载白名单 SP_SUPPORTED_ARCHITECTURES（当前仅 Qwen3ForCausalLM）、initialize_layernorm_sp（分布式初始化后按 worker 物化 flags.sp.enabled）、边界 collective sp_entry_scatter / sp_exit_gather、出口判定 maybe_exit_gather，以及基于 torch symmetric memory 的 fused matmul+collective 快路径 column_parallel_g_matmul / row_parallel_gbar_matmul（带普通 collective 回退）。关键设计：runs_sp(forward_mode) 只依赖配置与 forward 模式，专供 CUDA graph 捕获区域外使用；_SPForwardState.num_tokens 使用实例属性而非 ForwardFlags 整数槽，避免 torch.compile 按序列长度重新编译。
2. 接入前向执行链路：layers/communicator.py 的 LayerCommunicator 在 SP 启用时构造一个全 SCATTERED 的 _sp_variant sibling，并在 prepare_attn 中于首个 decoder layer（residual is None）判定 EXTEND 写入 sp_active、执行入口 scatter，随后 prepare_attn / prepare_mlp / postprocess_layer 全部委托给 sibling，使 norm/residual 在本地 shard 上计算、不触发额外 collective；layers/linear.py 的 ColumnParallelLinear

与 RowParallelLinear 在 sp_active 且 tp_size > 1 时分别进入 g（all-gather+matmul）与 g-bar（matmul+reduce-scatter）分支；layers/logits_processor.py 在 LM head 之前调用 maybe_exit_gather 撤销切分——判定用 runs_sp 而非 sp_active，因为捕获区域内写入的标志在 CUDA graph 回放时不会重执行。

3. 运行时状态与配置入口：runtime_context.py 新增 SpFlags（挂到 Flags 根下）与 ForwardFlags.sp_active 槽，并把它加入 _GRAPH_VISIBLE 保证图捕获期间线性层可读；server_args.py 新增 --enable-layernorm-sp（parallel 命名空间）；distributed/bootstrap.py 在 initialize_dp_attention 旁边调用 initialize_layernorm_sp。
4. 配置校验：新增 arg_groups/layernorm_sp_hook.py 并注册进解析管道（pipeline.py），validate_layernorm_sp 依次拒绝：白名单外架构、tp_size <= 1、--enable-dp-attention、投机解码；另有静默门控：仅 EXTEND 生效；fused 快路径要求未量化、无 bias 的 bf16/fp16 线性层，否则回退普通 collective。
5. 测试配套：CPU 单测 test/registered/unit/layers/test_layernorm_sp.py（白名单、prefill-only、四项校验、runs_sp 忽略 sp_active 的回归、sp_active 槽注册检查）；e2e test/registered/models_e2e/test_layernorm_sp.py 以 Qwen3-8B + --tp 2 --enable-layernorm-sp 跑 GSM8K（阈值 0.85），分别注册为 CPU 与 2-gpu-large CI 阶段。

关键文件：

- python/sglang/srt/layers/layernorm_sp.py（模块 序列并行；类别 source；类型 core-logic；符号 initialize_layernorm_sp, layernorm_sp_enabled, runs_sp, _SPForwardState）：新增 SP 核心模块：白名单、启用判定、entry/exit collective、fused matmul 快路径全部集中于此，是 models 零侵入解耦的关键。
- python/sglang/srt/arg_groups/layernorm_sp_hook.py（模块 配置校验；类别 source；类型 configuration；符号 handle_layernorm_sp, validate_layernorm_sp）：启动期校验 hook：在解析管道中拒绝不支持的架构、tp_size<=1、dp-attention 与投机解码，保证不兼容配置 fail loud。
- python/sglang/srt/layers/communicator.py（模块 层通信；类别 source；类型 core-logic）：LayerCommunicator 新增 _sp_variant 全 SCATTERED sibling 并委托 prepare_attn / prepare_mlp / postprocess_layer，是 SP 区域在解码器内运转的枢纽。
- python/sglang/srt/layers/linear.py（模块 线性层；类别 source；类型 dependency-wiring）：Row / ColumnParallelLinear 在 sp_active 时自动参与 g / g-bar，无需在模型代码里逐个标记，是零侵入解耦的关键。
- python/sglang/srt/runtime_context.py（模块 运行时状态；类别 source；类型 core-logic；符号 SpFlags）：新增 SpFlags 与 ForwardFlags.sp_active 槽（含 _GRAPH_VISIBLE 注册），是 SP 运行期状态的基础。
- python/sglang/srt/layers/logits_processor.py（模块 输出头；类别 source；类型 dependency-wiring）：LM head 前的 exit gather 接入点，调用 maybe_exit_gather 撤销序列切分，模型无关。
- python/sglang/srt/server_args.py（模块 参数配置；类别 source；类型 configuration）：新增 --enable-layernorm-sp 用户入口参数（parallel 命名空间），是功能对外开关。

- python/sglang/srt/distributed/bootstrap.py（模块 分布式启动；类别 source；类型 entrypoint）：分布式初始化后调用 initialize_layernorm_sp，物化 flags.sp.enabled，与 initialize_dp_attention 并列。
- python/sglang/srt/arg_groups/pipeline.py（模块 解析管线；类别 source；类型 entrypoint）：把 layernorm_sp_hook 注册进解析管道，使校验成为 resolution pipeline 的一部分。
- test/registered/unit/layers/test_layernorm_sp.py（模块 单元测试；类别 test；类型 test-coverage；符号 _initialize, TestLayerNormSPGating, tearDown, test_initialize_enables_only_for_allowlisted_arch）：CPU 单测覆盖白名单、prefill-only、四项校验与 CUDA graph 相关回归，锁死 runs_sp 语义。
- test/registered/models_e2e/test_layernorm_sp.py（模块 端到端测试；类别 test；类型 test-coverage；符号 TestLayerNormSPAccuracy, setUpClass, tearDownClass）：e2e 精度测试：Qwen3-8B tp=2 GSM8K 阈值 0.85，验证 SP 不回归模型输出。

关键符号：initialize_layernorm_sp, layernorm_sp_enabled, runs_sp, set_sp_num_tokens, sp_num_tokens, sp_entry_scatter, sp_exit_gather, maybe_exit_gather, sp_fused_matmul_eligible, column_parallel_g_matmul, row_parallel_gbar_matmul, handle_layernorm_sp, validate_layernorm_sp

关键源码片段

python/sglang/srt/layers/layernorm_sp.py

新增 SP 核心模块：白名单、启用判定、entry/exit collective、fused matmul 快路径全部集中于此，是 models 零侵入解耦的关键。

```
# Megatron 风格 LayerNorm 序列并行（arXiv:2205.05198）的边界与门控核心。

# SP 只对白名单架构生效；机制是通用的，后续模型只需扩展这一行并做硬件验证。
SP_SUPPORTED_ARCHITECTURES = frozenset({'Qwen3ForCausalLM'})


def runs_sp(forward_mode) -> bool:
    '''判断本次 forward 是否运行 SP：模型启用且仅限 prefill（EXTEND）。

    CUDA graph 捕获区域之外的代码必须用它，不能读 ``sp_active``——
    捕获区域内 Python 写入在 graph replay 时不会重执行，flag 是过期的。
    '''
    from sglang.srt.model_executor.forward_batch_info import ForwardMode

    return layernorm_sp_enabled() and forward_mode == ForwardMode.EXTEND


class _SPForwardState:
    '''保存当前 SP forward 的真实（未 padding）token 数。

    特意用实例属性而非 ForwardFlags 整数槽：该值在 torch.compile 追踪的
    线性层代码内被读取，dynamo 对属性来源 int 按 automatic-dynamic 处理，
```

```
    而 dict 槽 int 会随序列长度触发重新编译。
    '''

    num_tokens: int = 0


def sp_entry_scatter(hidden_states: torch.Tensor) -> torch.Tensor:
    '''把复制的 [M, h] hidden states 沿 token 维切分。

    M 会 pad 到 tp_size 整数倍，padding 行由出口侧丢弃；输入在 TP group 内
    各 rank 持有完整副本，所以这里只是本地切片，无需通信。
    '''
    num_tokens = hidden_states.shape[0]
    set_sp_num_tokens(num_tokens) # 记录真实 token 数，供 g 侧裁剪
    tp_group = get_tp_group()
    tp_size = tp_group.world_size
    if tp_size == 1:
        return hidden_states
    padded = ceil_align(num_tokens, tp_size)
    if padded != num_tokens:
        hidden_states = torch.nn.functional.pad(
            hidden_states, (0, 0, 0, padded - num_tokens)
        )
    return hidden_states.tensor_split(tp_size)[tp_group.rank_in_group].contiguous()


def sp_exit_gather(hidden_states: torch.Tensor, num_tokens: int) -> torch.Tensor:
    '''g：沿 dim 0 all-gather 各 rank 的 shard，再 narrow 到真实 token 数。'''
    tp_group = get_tp_group()
    tp_size = tp_group.world_size
    if tp_size == 1:
        return hidden_states[:num_tokens]
    output = hidden_states.new_empty(
        (hidden_states.shape[0] * tp_size, *hidden_states.shape[1:])
    )
    tp_group.all_gather_into_tensor(output, hidden_states.contiguous())
    return output[:num_tokens]


def maybe_exit_gather(*, hidden_states, hidden_states_before_norm, input_ids, forward_mode):
    '''LM head 之前撤销 sequence sharding 并离开 SP 区域。

    token 数取自 input_ids、判定用 runs_sp，而不是读 sp_active：CUDA graph
    回放时捕获区内写入不会重执行，读 flag 会把序列分片的 hidden states
    喂给 LM head 造成错误输出。
    '''
    if not runs_sp(forward_mode) or input_ids is None:
        return hidden_states, hidden_states_before_norm
    num_tokens = input_ids.shape[0]
```

```
    hidden_states = sp_exit_gather(hidden_states, num_tokens=num_tokens)
    if hidden_states_before_norm is not None:
        hidden_states_before_norm = sp_exit_gather(
            hidden_states_before_norm, num_tokens=num_tokens
        )
    get_forward().set('sp_active', False)
    return hidden_states, hidden_states_before_norm
```

python/sglang/srt/layers/communicator.py

LayerCommunicator 新增 _sp_variant 全 SCATTERED sibling 并委托 prepare_attn / prepare_mlp / postprocess_layer，是 SP 区域在解码器内运转的枢纽。

```
class LayerCommunicator:
    def __init__(self, ..., _is_sp_variant: bool = False):
        # SP 启用时，norm/residual 在序列 shard 上运行且不需要 collective，
        # 因此构造一个全 SCATTERED 的 sibling 通信器并委托给它；
        # _is_sp_variant 防止 sibling 递归再套一层。
        self._sp_variant: Optional[LayerCommunicator] = None
        if not _is_sp_variant and layernorm_sp.layernorm_sp_enabled():
            self._sp_variant = LayerCommunicator(
                layer_scatter_modes=LayerScatterModes(
                    layer_input_mode=ScatterMode.SCATTERED,
                    attn_mode=ScatterMode.SCATTERED,
                    mlp_mode=ScatterMode.SCATTERED,
                    middle_residual_mode=ScatterMode.SCATTERED,
                    layer_output_mode=ScatterMode.SCATTERED,
                ),
                input_layernorm=input_layernorm,
                post_attention_layernorm=post_attention_layernorm,
                allow_reduce_scatter=allow_reduce_scatter,
                is_last_layer=is_last_layer,
                qkv_latent_func=qkv_latent_func,
                force_layernorm_before_dp_gather=force_layernorm_before_dp_gather,
                enable_fused_ar_quant=enable_fused_ar_quant,
                fused_ar_quant_keep_bf16=fused_ar_quant_keep_bf16,
                _is_sp_variant=True,
            )

    def prepare_attn(self, hidden_states, residual, forward_batch, ...):
        # residual 为 None 标记首个 decoder layer，SP 区域在此打开；
        # 每个 forward 都重新判定，避免中途崩溃把状态泄漏到下一轮。
        if self._sp_variant is not None:
            if residual is None:
                get_forward().set(
                    'sp_active', forward_batch.forward_mode == ForwardMode.EXTEND
                )
                if get_forward().sp_active:
                    # 入口 scatter：此后各层看到的 token 数都是本 rank 的 1/tp 分片。
                    hidden_states = layernorm_sp.sp_entry_scatter(hidden_states)
```

```
            if get_forward().sp_active:
                # 委托给全 SCATTERED sibling：norm/residual 在本地 shard 上
                # 计算，不触发任何 gather/scatter collective。
                return self._sp_variant.prepare_attn(
                    hidden_states, residual, forward_batch, quant_format, post_residual_addition
                )
        # 非 SP：原有 input_scattered 判定与 norm 逻辑继续执行。
```

python/sglang/srt/layers/linear.py

Row / ColumnParallelLinear 在 sp_active 时自动参与 g / g-bar，无需在模型代码里逐个标记，是零侵入解耦的关键。

```
class ColumnParallelLinear(LinearBase):
    def forward(self, input_):
        bias = self.bias if not self.skip_bias_add else None

        # Megatron SP 的 g：输入是本 rank 的 [M_pad/tp, K] 序列分片，
        # 先 all-gather 回完整序列再 matmul。参与层（qkv / gate_up）的
        # gather_output=False，因此没有输出侧 all-gather 需要处理。
        if get_forward().sp_active and self.tp_size > 1:
            output = layernorm_sp.column_parallel_g_matmul(self, input_, bias)
            output_bias = self.bias if self.skip_bias_add else None
            return output, output_bias

        # 非 SP 普通路径：量化方法 matmul + 可选输出 all-gather。
        output_parallel = self.quant_method.apply(self, input_, bias)
        if self.gather_output:
            output = tensor_model_parallel_all_gather(output_parallel)
        else:
            output = output_parallel
        output_bias = self.bias if self.skip_bias_add else None
        return output, output_bias


class RowParallelLinear(LinearBase):
    def forward(self, input_, skip_all_reduce=False, ...):
        # bias 只在 rank 0 参与 GEMM，避免 TP>1 时被重复累加。
        bias_ = None if (self.tp_rank > 0 or self.skip_bias_add) else self.bias

        # Megatron SP 的 g-bar：沿 token 维 reduce-scatter 替代 all-reduce，
        # 输出保持分片供下一段 SP LayerNorm 使用。o_proj / down 的
        # reduce_results=False，因此 SP 下由线性层自己承担归约。
        if (
            get_forward().sp_active
            and self.tp_size > 1
            and not skip_all_reduce
            and output_tensor is None
        ):
            output = layernorm_sp.row_parallel_gbar_matmul(self, input_parallel, bias_)
```

```
            output_bias = self.bias if self.skip_bias_add else None
            return output, output_bias

        # 非 SP 路径：原 reduce / all-reduce 逻辑继续执行。
```

评论区精华

review 的核心交锋集中在架构收敛、模型解耦与测试配套：

- SP 逻辑收敛：Fridge003 要求 "Can we open a single file for layernorm_sp, and put all the related classes/util functions there... The dp_attention.py file shouldn't be modified"；MartinHua 回复 "Everything SP now lives in layernorm_sp.py... dp_attention.py is fully reverted — zero changes"。
- 模型零侵入："Ideally there should be no change in files under python/sglang/srt/models"——最终通过 LayerCommunicator 委托 + 线性层自动参与实现，models/ 下零改动，新模型启用只需 allowlist 一行加硬件验证。
- 测试要求："Please add a unit test and an e2e test for layernorm sp"、"Please make it a cpu test"、"Please use Qwen3-8B, and 0.85 as gsm8k score threshold"——全部落地，单测 9 项全过、e2e 约 67 秒。
- 校验位置："Please create a layernorm_sp_hook under args_group, and move the validation there. The validation should be a part of _run_resolution_pipeline"——新增 arg_groups/layernorm_sp_hook.py，四项校验在启动期 fail loud。
- 冗余与注释："This function looks redundant" / "This function is also redundant. We can use inline get_forward().sp_active instead"——enabled 迁入 SpFlags；runs_sp 保留并新增 test_runs_sp_ignores_the_active_flag 回归测试固化语义（CUDA graph replay 下 flag 过期）；AI 注释按要求清理。
- 越界改动："Don't change this file"（benchmark/one_batch.py）——已回退。
  - SP 逻辑收敛到独立模块 layernorm_sp.py (design): 全部 SP 逻辑（白名单、校验、sp_active 标志、entry/exit collective、fused g/g-bar helper）集中在 layers/layernorm_sp.py，dp_attention.py 无任何变更。
  - 模型代码零侵入解耦 (design): models/ 下零改动；新模型启用只需 allowlist 一行加硬件验证，校验放在解析管道中 fail loud。
  - 单元测试与端到端测试配套 (testing): 落地 CPU 单测与 2-gpu-large e2e（Qwen3-8B、tp=2、GSM8K 阈值 0.85），并注册进 CI（stage base-a / base-b）。
  - 校验逻辑移入 arg_groups 解析管道 (design): 新增 arg_groups/layernorm_sp_hook.py，校验覆盖架构白名单、tp_size、dp-attention、投机解码四项。
  - 冗余函数与 AI 注释清理 (style): enabled 迁入 runtime_context.SpFlags；runs_sp 保留并新增 test_runs_sp_ignores_the_active_flag 回归用例；注释精简。
  - 保持 benchmark/one_batch.py 不被改动 (other): 该文件最终不在变更集中。

风险与影响

- 风险：

1. CUDA graph 状态一致性：sp_active 在捕获区内写入、回放时过期，已用 runs_sp 规避并有回归测试兜底；但 _SPForwardState.num_tokens 是模块级全局实例，若 forward 中途异常退出，可能把上一轮 token 数泄漏到下一 forward（代码通过每 forward 重写缓解该风险）。
2. 依赖 torch 内部 API：fused 快路径探测 torch.ops.symm_mem.fused_matmul_reduce_scatter / fused_all_gather_matmul，torch 升级可能改变这些内部符号；有探测 +fallback 缓解，但 fallback 会丢失 B200 上的主要收益。
3. 覆盖范围窄：白名单仅 Qwen3ForCausalLM，e2e 只跑 Qwen3-8B / tp=2；padding 非对齐、更大 TP、B200 fused path 的正确性主要依赖手动验证。
4. NVLink 前置未强制：PR body 明说 "NVLink/NVSwitch is required in practice but not yet enforced"，非 NVLink 环境开启可能无收益甚至负收益。
5. 默认路径影响小：默认关闭且分支由 sp_active 门控，普通用户无行为变化；关闭时字节一致。 - 影响：用户侧：默认关闭，行为零变化；开启后仅影响 Qwen3 dense + tp>1 的 prefill，长上下文收益显著（激活内存降 27.7% @ 8×H100），精度无回归。系统侧：核心执行路径（linear / communicator / logits_processor）新增 SP 分支，但由 sp_active 门控，默认路径计算开销几乎为零。团队侧：建立了可扩展的 SP 接入模板——新模型家族只需 allowlist 一行 + 硬件验证；PR body 已列出 B300、DP-attention 感知 SP、投机解码、chunked/split prefill 等后续方向。 - 风险标记：CUDA graph 标志过期风险，依赖 torch 内部 symm_mem API，仅 Qwen3 白名单，NVLink 前置未强制，异常路径状态泄漏

关联脉络

- PR #36933 [2/N][Mixed] Mixed chunk prefill with spec enabled: 同为调度 / 前向路径改造：SP 的 prefill-only 判定基于 forward_mode == EXTEND，mixed chunk prefill 扩展了 forward 模式组合，二者在 runs_sp 判定与 EXTEND 语义上存在潜在交互，需要后续回归验证。