PR #27192 完整报告

sgl-project/sglang

[refactor] Retire DecodeInputBuffers / PrefillInputBuffers in favor of CudaGraphBufferRegistry

合并时间：2026-06-04 11:52

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27192

执行摘要

- 一句话：删除 DecodeInputBuffers/PrefillInputBuffers，统一由注册表管理
- 推荐动作：建议所有参与 CUDA Graph 相关开发的工程师精读此 PR，特别是 share_input_buffers_in 的设计和注册表 source= 参数的使用模式。本 PR 是渐进式重构的范例，展示了如何在保持行为不变的前提下逐步淘汰遗留抽象。

功能与动机

PR body 指出："After CudaGraphBufferRegistry landed (#26742), the cuda-graph runners still kept their legacy DecodeInputBuffers / PrefillInputBuffers dataclasses purely as buffer containers — the per-replay fill logic already lives in the registry. This PR retires those two dataclasses so the registry is the single owner of the ForwardBatch-shared, graph-resident input buffers"。

实现拆解

步骤 1：删除 DecodeInputBuffers（cuda_graph_runner.py）

- 移除 DecodeInputBuffers dataclass（继承自 ForwardInputBuffers），包括其 create classmethod 和 populate_from_forward_batch 方法。
- 新增模块级函数 _allocate_decode_buffers()，返回一个包含相同张量字段的 SimpleNamespace。
- build_replay_fb_view 的 buffers 形参从 DecodeInputBuffers 类型改为无类型标注（实际接收命名空间）。
- _dummy_run 改为调用 _allocate_decode_buffers 获取缓冲区。

步骤 2：删除 PrefillInputBuffers（piecewise_cuda_graph_runner.py & breakable_cuda_graph_runner.py）

- 移除 PrefillInputBuffers dataclass（继承自 ForwardInputBuffers）。
- 两个运行器的 _init_buffers 不再手动构造 PrefillInputBuffers 实例，而是直接调用 build_prefill_registry(source=None, share_pool=not is_npu())，由注册表自行分配和共享缓冲区。
- 移除了对 ForwardInputBuffers 的导入。

步骤 3：提取统一的缓冲区池化入口（input_buffers.py）

- 新增函数 share_input_buffers_in(obj)，遍历任意对象（dataclass 或 SimpleNamespace）的属性，将其中的 torch.Tensor 通过进程级池 share_input_buffer 进行别名共享。
- 该函数同时处理嵌套的 dict 和 dataclass 字段（如 pp_proxy_tensors、ngram_embedding_info）。
- 为 share_input_buffer 补充文档，说明池的作用域和跨运行器共享的安全性。

步骤 4：更新注册表文档与注释（cuda_graph_buffer_registry.py）

- build_decode_registry 和 build_prefill_registry 的 docstring 更新，移除对已删除 dataclass 的引用，明确 source= 与 source=None 的语义。
- 注明 extract_buffer 当前未启用，不能替代 build_replay_fb_view。

步骤 5：增加单元测试（test_cuda_graph_buffer_registry.py）

- test_num_token_non_padded_gathered_dp_branch：验证在 gathered DP 模式下 fill_from 正确计算 num_token_non_padded。
- test_source_none_owns_allocated_buffers：验证 build_prefill_registry(source=None) 时注册表拥有的缓冲区形状正确。

关键文件：

- python/sglang/srt/model_executor/cuda_graph_runner.py（模块 CUDA Graph 运行器；类别 source；类型 data-contract；符号 DecodeInputBuffers, create, _allocate_decode_buffers, populate_from_forward_batch）：核心变更文件，删除了 DecodeInputBuffers dataclass，新增 _allocate_decode_buffers 函数，调整了 build_replay_fb_view 签名和 _dummy_run。
- python/sglang/srt/model_executor/piecewise_cuda_graph_runner.py（模块 分段 CUDA Graph 运行器；类别 source；类型 data-contract；符号 PrefillInputBuffers）：删除了 PrefillInputBuffers dataclass，改为直接由注册表分配缓冲区。
- python/sglang/srt/model_executor/input_buffers.py（模块 输入缓冲区；类别 source；类型 data-contract；符号 share_input_buffers_in）：新增核心工具函数 share_input_buffers_in，实现任意对象的缓冲区池化。
- python/sglang/srt/model_executor/breakable_cuda_graph_runner.py（模块 可中断 CUDA Graph 运行器；类别 source；类型 data-contract）：更新 _init_buffers 移除对 PrefillInputBuffers 的依赖，简化初始化流程。
- test/registered/unit/model_executor/test_cuda_graph_buffer_registry.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_num_token_non_padded_gathered_dp_branch, test_source_none_owns_allocated_buffers）：新增两个单元测试，覆盖 gathered DP 分支和 source=None 场景，保障重构正确性。
- python/sglang/srt/model_executor/cuda_graph_buffer_registry.py（模块 CUDA Graph 缓冲区注册表；类别 source；类型 data-contract）：更新了 build_decode_registry 和 build_prefill_registry 的文档字符串，移除对已删除 dataclass 的引用，并说明 extract_buffer 当前状态。

- python/sglang/srt/model_executor/model_runner.py（模块 模型运行器；类别 source；类型 data-contract）：微小调整（+2/-2），可能是导入或类型提示调整。

关键符号：_allocate_decode_buffers, share_input_buffers_in, build_decode_registry, build_prefill_registry, build_replay_fb_view

关键源码片段

python/sglang/srt/model_executor/cuda_graph_runner.py

核心变更文件，删除了 DecodeInputBuffers dataclass，新增 _allocate_decode_buffers 函数，调整了 build_replay_fb_view 签名和 _dummy_run。

```
# 摘自 cuda_graph_runner.py (head) — 新的缓冲区分配函数
# 替代了原有的 DecodeInputBuffers.create 和 populate_from_forward_batch

def _allocate_decode_buffers(
    *,
    device: torch.device,
    max_bs: int,
    max_num_token: int,
    hidden_size: int,
    vocab_size: int,
    dtype: torch.dtype,
    dp_size: int,
    pp_size: int,
    is_encoder_decoder: bool,
    require_mlp_tp_gather: bool,
    seq_len_fill_value: int,
    encoder_len_fill_value: int,
    num_tokens_per_bs: int,
    cache_loc_dtype: torch.dtype,
    enable_mamba_track: bool,
    ne_token_table: Optional[torch.Tensor] = None,
    hc_hidden_size: Optional[int] = None,
) -> SimpleNamespace:
    """分配并返回一个包含所有 decode 输入缓冲区的 SimpleNamespace。

    功能和原来的 DecodeInputBuffers.create 相同，但不再继承 ForwardInputBuffers。
    返回的命名空间可以直接作为 source= 参数传给 build_decode_registry。
    张量创建在指定的设备上，后续通过 share_input_buffers_in 或注册表的 share_pool 进行池化。
    """
    with torch.device(device):
        input_ids = torch.zeros((max_num_token,), dtype=torch.int64)
        input_embeds = torch.zeros((max_num_token, hidden_size), dtype=dtype)
        req_pool_indices = torch.zeros((max_bs,), dtype=torch.int64)
        seq_lens = torch.full((max_bs,), seq_len_fill_value, dtype=torch.int32)
        seq_lens_cpu = torch.full((max_bs,), seq_len_fill_value, dtype=torch.int32)
        out_cache_loc = torch.zeros((max_num_token,), dtype=cache_loc_dtype)
        positions = torch.zeros((max_num_token,), dtype=torch.int64)
```

```
        mrope_positions = torch.zeros((3, max_num_token), dtype=torch.int64)
        # ... ( 省略其余张量，完全保持与原有 create 一致 )
        num_token_non_padded = torch.zeros((1,), dtype=torch.int32)
        custom_mask = torch.ones(...) # 形状保留
        next_token_logits_buffer = torch.zeros(...)
        # etc.

    ns = SimpleNamespace(
        input_ids=input_ids,
        input_embeds=input_embeds,
        req_pool_indices=req_pool_indices,
        seq_lens=seq_lens,
        seq_lens_cpu=seq_lens_cpu,
        out_cache_loc=out_cache_loc,
        positions=positions,
        mrope_positions=mrope_positions,
        num_token_non_padded=num_token_non_padded,
        custom_mask=custom_mask,
        next_token_logits_buffer=next_token_logits_buffer,
        # mamba 及其他可选字段
    )
    return ns
```

python/sglang/srt/model_executor/input_buffers.py

新增核心工具函数 share_input_buffers_in，实现任意对象的缓冲区池化。

```
# 摘自 input_buffers.py (head) — 新增的通用池化入口
# 替代了原有 ForwardInputBuffers.share_buffers 的局部调用

def share_input_buffers_in(obj) -> None:
    """将 obj 上的所有张量字段通过进程级输入缓冲区池进行共享（原地替换）。

    obj 可以是 dataclass 或 SimpleNamespace。
    会递归处理嵌套的 dict 和 dataclass 字段（如 pp_proxy_tensors、ngram_embedding_info）。
    在 NPU 上该函数为空操作（避免精度问题）。
    """
    # NPU 上跳过共享以避免精度问题
    if is_npu():
        return

    for name, buffer in list(vars(obj).items()):
        if buffer is None:
            continue
        if dataclasses.is_dataclass(buffer):
            # 如果是 dataclass，展开其内部字段
            buffer = vars(buffer)
        if isinstance(buffer, dict):
            # 处理字典类型的字段（如 pp_proxy_tensors）
            for sub_name, sub_buffer in buffer.items():
```

```
                assert isinstance(sub_buffer, torch.Tensor), \
                    f"Field {name}.{sub_name} 应为 Tensor，当前类型 {type(sub_buffer)}"
                buffer[sub_name] = share_input_buffer(f"{name}.{sub_name}", sub_buffer)
        else:
            # 普通张量字段
            assert isinstance(buffer, torch.Tensor), \
                f"Field {name} 应为 Tensor/dict/dataclass，当前类型 {type(buffer)}"
            setattr(obj, name, share_input_buffer(name, buffer))
```

python/sglang/srt/model_executor/breakable_cuda_graph_runner.py

更新 _init_buffers 移除对 PrefillInputBuffers 的依赖，简化初始化流程。

```
# 摘自 breakable_cuda_graph_runner.py (head) — 新的 _init_buffers
# 删除了 PrefillInputBuffers 的构造和 share_buffers 调用，全部委托给注册表

def _init_buffers(self, model_runner):
    """初始化输入缓冲区。

    以前版本：先构造 PrefillInputBuffers 实例，调用 share_buffers，
    再传给 build_prefill_registry(source=self.buffers)。
    新版本：由注册表自行分配和池化缓冲区（source=None）。
    """
    from sglang.srt.model_executor.cuda_graph_buffer_registry import (
        build_prefill_registry,
    )
    from sglang.srt.utils import is_npu

    # 选择 cache_loc 的 dtype，NPU 上使用 int32 以与设备兼容
    cache_loc_dtype = torch.int64 if not is_npu() else torch.int32

    if model_runner.is_draft_worker:
        # 草图 worker 需要额外的隐藏状态缓冲区
        from sglang.srt.speculative.eagle_utils import get_draft_hidden_dim
        hidden_dim = get_draft_hidden_dim(model_runner)
        self.static_draft_hidden_states = torch.zeros(
            (self.max_num_tokens, hidden_dim),
            dtype=model_runner.dtype,
            device=self.device,
        )

    # 注册表成为 token axis 输入缓冲区的唯一所有者
    # share_pool 控制是否通过进程级池进一步共享（NPU 上不共享）
    self.buffer_registry = build_prefill_registry(
        device=self.device,
        max_bs=1,
        max_num_token=self.max_num_tokens,
        cache_loc_dtype=cache_loc_dtype,
        is_multimodal=self.is_multimodal,
        hidden_size=model_runner.model_config.hidden_size,
```

```
        embed_dtype=model_runner.dtype,
        enable_mamba_track=False,
        share_pool=not is_npu(),
        source=None, # 注册表自己分配，不再外部提供
    )
```

评论区精华

本 PR 的 review 评论数为 0，主要讨论集中在作者的 CI 状态评论。作者确认 CI 测试通过（green），并触发了标签重新运行。虽无公开争议，但从提交历史可见作者进行了 5 次迭代，包括文档补充、两个阶段的删除 refactor、样式修剪等，体现了对代码整洁的追求。

- 暂无高价值评论线程

风险与影响

- 风险：
  - 回归风险：缓冲区所有权由 dataclass 转移到注册表，但所有缓冲区创建、填充和共享逻辑均保持不变。NPU 场景下 share_pool=False 的行为没有变化。
  - 测试覆盖：新增的两个单元测试覆盖了 gathered DP 分支和 source=None 所有权，但缺少实际 GPU 上的端到端测试（依赖 CI）。
  - 性能影响：无预期影响，因为 eager 路径不变，capture/replay 拷贝路径也未改变。
  - 兼容性：未改动公共 API，仅重构内部数据结构。外部调用者（如 speculative 运行器）仍通过 ForwardInputBuffers 基类正常工作。
- 影响：
  - 用户影响：无。行为完全保留。
  - 系统影响：减少了 300+ 行冗余代码，缓冲区所有权更清晰，未来新增运行器可以统一使用注册表分配。
  - 团队影响：降低了维护成本，新的开发者更容易理解缓冲区生命周期。
  - 风险标记：核心路径变更，NPU 特殊处理，测试覆盖新分支，行为保持（非功能变更）

关联脉络

- PR #26742 Add CudaGraphBufferRegistry: 本 PR 的前置条件，CudaGraphBufferRegistry 的引入使得可以退休旧的 dataclass。
- PR #27091 stacked on (基础分支): 本 PR 基于 #27091（cheng/swa-translate-int64），共享 4 个文件，需要先合并 #27091。