PR #33949 完整报告

sgl-project/sglang

fix(vlm): stream-order CUDA IPC feature pool lifecycle and streamline multimodal transport module

合并时间：2026-08-10 18:47

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33949

执行摘要

- 一句话：流序化 CUDA IPC 特征池生命周期并重构传输模块
- 推荐动作：值得精读，尤其适合关注跨进程 GPU 传输与 CUDA 流同步的读者：generation 标记 + 流序 ack 的池设计、句柄缓存失效回退策略、以及 encoder-DP 下“延迟物化到持有者 rank”的回调模式都具备复用价值。建议先读 memory_pool.py 的协议定义，再对照 cuda_ipc.py 的 proxy 重建与 qwen3_vl.py 的 DP 物化分支，最后看 test_mm_process_config.py 中针对协议边界的单测。

功能与动机

PR 正文指出旧协议的风险：“The old producer could serialize a proxy immediately after enqueueing its pool copy, while the consumer incremented a CPU acknowledgement immediately after enqueueing its local copy. That could expose incomplete bytes or recycle and overwrite a slice before the copy completed.” 即 CPU 计数无法与 GPU 异步拷贝对齐，导致消费者可能读到未拷贝完成的数据，或回收器提前复用一个仍在被拷贝读取的切片；为此引入基于 CUDA 流序和 generation 标记的 ready/wait/ack 协议。

实现拆解

整个过程分四步：

1. 新增流序池内核（memory_pool.py，+367 行）：新建 StreamOrderedMmFeaturePool、StreamOrderedPoolConsumerMixin、PoolLease 与 resolve_consumer_rank。池头部按 1 + consumer_count 个 32 位控制字为每个 in-flight 槽位布局，首个字是 producer 的 ready 标记，其余为各消费者 ack 字；stream_wait_value32 / stream_write_value32 通过 CUDA 驱动调用把同步挂在当前流上。槽位分配使用 generation 递增标记，_recycle_ready_leases_locked 用 torch.index_select 批量比对控制字与 generation，只有全员 ack 的租约才被回收；回收器线程独立轮询，并在 shutdown 时通过 stop event 及时退出。
2. 重组传输模块（cuda_ipc.py 新增，utils/cuda_ipc_transport_utils.py 由 568 行缩减为 24 行兼容层）：MmItemMemoryPool 和 CudaIpcTensorTransportProxy 从 utils 迁入 sglang/srt/multimodal/transport/cuda_ipc.py，处理器清理了每切片 POSIX 共享内存与全局文件锁；_pool_handle_cache_get_or_open 等句柄缓存逻辑保留，SGLANG_USE_IPC_POOL_HANDLE_CACHE 仍默认开启。un-cached 映射通过

_retain_storage_until_stream_completes 在释放前 torch.cuda.current_stream().synchronize()，句柄缓存打开失败时先失效缓存再走 un-cached 重试。

3. 接入点改造（base_processor.py、schedule_batch.py、kimi_k3/k25）：_wrap_tensor_for_cuda_ipc 改用 pool.wrap_tensor(tensor, use_pool_handle_cache=...)，MmItemMemoryPool 构造时传入 server_args.tp_size 作为 consumer_count；schedule_batch 与 Kimi 系列改为从新 transport 模块导入。
4. Qwen3-VL encoder-DP 延迟物化（qwen3_vl.py、processors/qwen_vl.py）：新增 _mark_dp_encoder_features_for_deferred_reconstruction，仅在 keep_mm_features_on_device && mm_enable_dp_encoder && qwen3_vl/qwen3_vl_moe/qwen3_5/qwen3_5_moe 时给 image/video item 打 DEFER_CUDA_IPC_FEATURE_RECONSTRUCTION_KEY 标记；模型侧 _get_visual_feature 在 DP 分支把 pixel values 物化推迟到 run_dp_sharded_mrope_vision_model 的 load_local_pixel_values 回调中，只重构本 rank 负责的 item；所有 CUDA 重构路径都会以 max(tp_size, 1) 作为 ipc_consumer_count。
5. 测试与基准配套：新增 TestStreamOrderedMmFeaturePool（generation 复用、group ack 地址、重复 release 拒绝、recycler 关闭），扩展 qwen3-vl 特征物化回归（DP 延迟、本地 item 物化、reconstruct 调用次数），spawned-process CUDA IPC 回归不再同步 producer 并验证回收；H200 x8 真实权重 A/B 无失败请求。

关键文件：

- python/sglang/srt/multimodal/transport/memory_pool.py（模块 传输池；类别 source；类型 core-logic；符号 align_up, _driver_modules, stream_wait_value32, stream_write_value32）：新增的流序内存池核心：ready/wait/ack 控制字、generation 槽位复用、回收器线程，是整个 PR 正确性修复的基石。
- python/sglang/srt/multimodal/transport/cuda_ipc.py（模块 传输层；类别 source；类型 core-logic；符号 get_mm_feature_pool_size_per_worker, _normalize_pool_cache_key, _open_pooled_storage_uncached, _pool_handle_cache_get_or_open）：CUDA IPC 传输的新家：MmItemMemoryPool 包装流序池，CudaIpcTensorTransportProxy 实现句柄缓存、un-cached 生命周期与重建；旧模块改为兼容导入。
- python/sglang/srt/utils/cuda_ipc_transport_utils.py（模块 兼容层；类别 source；类型 refactor；符号 get_mm_feature_pool_size_per_worker, _normalize_pool_cache_key, _open_pooled_storage_uncached, _pool_handle_cache_get_or_open）：568 行实现被压缩为 24 行兼容导入层，是模块重构的落地证据；ShmSyncBuffer、文件锁与切片回收逻辑全部移除。
- python/sglang/srt/models/qwen3_vl.py（模块 模型层；类别 source；类型 data-contract；符号 get_image_feature, get_video_feature, _get_visual_feature, _materialize_visual_items）：图像 / 视频特征物化重构：DP 编码器路径不再预取全部 pixel values，改为 load_local_pixel_values 回调按持有者 rank 物化，并传递 ipc_consumer_count。

- python/sglang/srt/multimodal/processors/qwen_vl.py（模块 处理器；类别 source；类型 core-logic；符号 _mark_dp_encoder_features_for_deferred_reconstruction）：新增 DP 编码器场景下延迟 CUDA IPC 特征重建的标记逻辑，并接入 DEFER_CUDA_IPC_FEATURE_RECONSTRUCTION_KEY。
- python/sglang/srt/multimodal/processors/base_processor.py（模块 处理器；类别 source；类型 dependency-wiring）：接入新池 API：MmItemMemoryPool 构造增加 consumer_count（tp_size），_wrap_tensor_for_cuda_ipc 改为 wrap_tensor 并显式传 use_pool_handle_cache。
- test/registered/unit/managers/test_mm_process_config.py（模块 测试；类别 test；类型 test-coverage；符号 TestStreamOrderedMmFeaturePool, test_consumer_slot_uses_global_tp_rank, test_complete_group_acknowledges_each_consumer_slot, test_reused_pool_slot_gets_new_generation）：新增 TestStreamOrderedMmFeaturePool，覆盖全局 rank 选择、group ack 地址、generation 复用、重复 release 拒绝、recycler 关闭。
- test/registered/unit/models/test_qwen3_vl_feature_materialization.py（模块 测试；类别 test；类型 test-coverage；符号 _model, test_processor_defers_gpu_transport_for_encoder_dp, test_processor_does_not_defer_cpu_transport, test_encoder_dp_materializes_only_locally_assigned_visual_items）：覆盖 processor 延迟 GPU transport、CPU transport 不延迟、encoder-DP 只重建本地 item 三条行为契约。
- test/registered/unit/multimodal/test_cuda_ipc_transport.py（模块 测试；类别 test；类型 test-coverage；符号 test_uncached_mapping_waits_before_proxy_release）：spawned-process CUDA IPC 回归改为不同步 producer，验证流序回收；新增 un-cached 映射释放前的流同步测试。
- python/sglang/srt/managers/schedule_batch.py（模块 调度器；类别 source；类型 dependency-wiring）：导入路径切换到新 transport 模块，保持 DEFER_CUDA_IPC_FEATURE_RECONSTRUCTION_KEY 与 CudaIpcTensorTransportProxy 的调度侧可见性。

关键符号：stream_wait_value32, stream_write_value32, resolve_consumer_rank, StreamOrderedPoolConsumerMixin._wait_until_ready, StreamOrderedPoolConsumerMixin._acknowledge_on_stream, StreamOrderedMmFeaturePool._allocate_locked, StreamOrderedMmFeaturePool._recycle_ready_leases_locked, StreamOrderedMmFeaturePool.shutdown, MmItemMemoryPool.wrap_tensor, CudaIpcTensorTransportProxy._open_pool_slice, CudaIpcTensorTransportProxy._retain_storage_until_stream_completes, Qwen3VLForConditionalGeneration._get_visual_feature, Qwen3VLForConditionalGeneration._materialize_visual_items, QwenVLImageProcessor._mark_dp_encoder_features_for_deferred_reconstruction, BaseMultimodalProcessor._wrap_tensor_for_cuda_ipc

关键源码片段

python/sglang/srt/models/qwen3_vl.py

图像 / 视频特征物化重构：DP 编码器路径不再预取全部 pixel values，改为 load_local_pixel_values 回调按持有者 rank 物化，并传递 ipc_consumer_count。

```
    def get_image_feature(self, items: List[MultimodalDataItem]) -> torch.Tensor:
        _require_vision(self)
        image_grid_thw = torch.concat([item.image_grid_thw for item in items], dim=0)
        # 图片和视频现在走同一套物化逻辑，差异只在 grid_thw 的来源。
        return self._get_visual_feature(items, image_grid_thw)

    def get_video_feature(self, items: List[MultimodalDataItem]) -> torch.Tensor:
        _require_vision(self)
        video_grid_thw = torch.concat([item.video_grid_thw for item in items], dim=0)
        return self._get_visual_feature(items, video_grid_thw)

    def _get_visual_feature(
        self, items: List[MultimodalDataItem], grid_thw: torch.Tensor
    ) -> torch.Tensor:
        assert grid_thw.dim() == 2, grid_thw.dim()
        if self.use_data_parallel:
            # encoder-DP 下不提前物化任何特征，把物化动作下沉到
            # run_dp_sharded_mrope_vision_model 的 load_local_pixel_values 回调里，
            # 只重建本 rank 分到的 visual items，避免每个 rank 都做完整 IPC 重建。
            return run_dp_sharded_mrope_vision_model(
                self.visual,
                None,
                grid_thw.tolist(),
                rope_type="rope_3d",
                load_local_pixel_values=partial(self._materialize_visual_items, items),
                pixel_values_device=self.visual.device,
                pixel_values_dtype=self.visual.dtype,
            )
        pixel_values = self._materialize_visual_items(items, range(len(items)))
        assert pixel_values.dim() == 2, pixel_values.dim()
        return self.visual(pixel_values, grid_thw=grid_thw)

    def _materialize_visual_items(
        self, items: List[MultimodalDataItem], indices: Iterable[int]
    ) -> torch.Tensor:
        device = self.visual.device
        device_index = device.index
        if device.type == "cuda" and device_index is None:
            device_index = torch.cuda.current_device()
        if device.type == "cuda":
            parallel = get_parallel()
            # ack 槽数必须等于全局 TP 大小，保证每个消费 rank 都有独立确认位
            consumer_count = max(parallel.tp_size, 1)

        features = []
```

```
        for index in indices:
            item = items[index]
            if device.type == "cuda":
                item.reconstruct(device_index, ipc_consumer_count=consumer_count)
            features.append(item.feature)
        return materialize_multimodal_features(
            features, device=device, dtype=self.visual.dtype
        )
```

评论区精华

该 PR 没有 reviewer 评论（review_comments_count = 0），仅有作者触发的 CI 命令与 Mintlify 文档预览。核心设计取舍记录在 PR 正文中：默认的 cached-handle 路径全程保持流序；当 handle caching 显式关闭时只同步消费者流再释放 un-cached 映射；放弃扩展自定义 FABRIC 传输，把多节点传输交给 #33899/#33936 的 CUDA VMM。作者还专门解释了 unlimited 请求率下 base/head 数值的差异来自新形状预热，并用第二轮饱和对比（64 req/run、3 组种子取中位数）证明稳态吞吐 +4.4%。

- 暂无高价值评论线程

风险与影响

- 风险：主要风险集中在三处：一是 stream_wait_value32 / stream_write_value32 依赖 CUDA 驱动级 API 与 vmm_utils.check_drv，在不支持该特性的驱动或非 CUDA 后端上会直接失败，需要确认旧平台兼容性；二是进程间协议变更，resolve_consumer_rank 使用全局 tp_rank 而非 attn_tp_rank，若存在 attention/DCP 子组别名时 ack 槽可能错位（测试 test_consumer_slot_uses_global_tp_rank 显式锁定了该选择，但仍需在真实子组配置下验证）；三是 Qwen3-VL DP 路径把特征物化回调化，run_dp_sharded_mrope_vision_model 的接口契约变化（新增 load_local_pixel_values、pixel_values_device、pixel_values_dtype）若与其他并行后端交互可能有兼容隐患。另外，旧模块被缩减为兼容导入层，任何直接依赖 cuda_ipc_transport_utils.Sh mSyncBuffer 等旧符号的第三方代码将失效。
- 影响：影响范围覆盖所有走 CUDA IPC 多模态特征传输的场景：tokenizer workers 与 TP 消费者之间的 GPU 张量传递、Qwen3-VL / Kimi-K3 / Kimi-K25 的图片和视频特征路径、encoder-DP 下的局部物化逻辑。默认开启且无需用户改配置（SGLANG_MM_FEATURE_CACHE_MB 仍是硬性 HBM 预算，SGLANG_USE_IPC_POOL_HANDLE_CACHE 默认开启，池压力仍回退 CPU），对部署方基本透明；对团队而言，模块收敛到 multimodal/transport 后，后续 CUDA VMM 多节点传输可复用统一控制字抽象，降低双协议维护成本。
- 风险标记：核心请求路径变更，依赖 CUDA 驱动流同步 API，进程间 IPC 协议变更，跨 18 文件大范围重构，外部导入兼容性风险

关联脉络

- PR #30392 [FEAT] Decouple multimodal global cache from Mooncake: 同属多模态特征缓存 / 传输解耦方向，本 PR 进一步把 CUDA IPC 传输收敛到独立模块，延续可插拔传输

后端的演进。

- PR #34163 fix(vlm): preserve Kimi-K3 GPU JPEG accuracy: 本 PR 改动了 kimi_k3.py / kimi_k25.py 及对应处理器 / 测试的导入路径，与 Kimi 系列 GPU 多模态路径直接相关。
- PR #34217 [misc] Pass FP8 scales in FlashInfer SWA prefill, autotune fp8 on SM120, and tighten is_image_understandable_model: 与本 PR 都改动了 model_config / 模型侧多模态相关判断，属于多模态模型能力收口的一部分。