PR #25895 完整报告

sgl-project/sglang

[Diffusion][NPU] Disaggregation diffusion stages support for NPU

合并时间：2026-05-25 18:51

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25895

执行摘要

- 一句话：NPU 扩散分解功能启用，抽象平台 API
- 推荐动作：值得精读。该 PR 清晰地展示了如何将 CUDA 硬编码代码迁移为平台无关的抽象，为后续支持更多硬件奠定了基础。current_platform 和 torch.get_device_module() 的使用模式值得作为跨平台开发的参考。

功能与动机

确保扩散模型分解功能在 Ascend NPU 上可用，消除对 CUDA 的硬编码依赖，提升平台可移植性。

实现拆解

1. 导入平台抽象：在 scheduler_mixin.py 中添加 from sglang.multimodal_gen.runtime.platforms import current_platform，为后续替换设备类型做准备。
2. 替换设备字符串：在 scheduler_mixin.py 中，将所有 torch.device(f"cuda:{local_rank}") 替换为 torch.device(f"{current_platform.device_type}:{local_rank}")，涉及 _init_disagg_request_scheduler、_init_disagg_state、_init_disagg_transfer_manager、_prefetch_transfer_ready、_broadcast_req_to_all_ranks、_disagg_prefetch_event_loop 六个函数。
3. 替换 Stream/Event API：在 manager.py 和 buffer.py 中，将 torch.cuda.Stream 替换为 torch.Stream，torch.cuda.Event 替换为 torch.Event，torch.cuda.stream(...) 替换为 torch.get_device_module().stream(...)，torch.cuda.current_stream() 替换为 torch.get_device_module().current_stream()，torch.cuda.is_available() 替换为 torch.get_device_module().is_available()，torch.cuda.synchronize() 替换为 torch.get_device_module().synchronize()。同时在 manager.py 中添加 current_platform 导入并将默认 device 从 "cuda" 改为 current_platform.device_type。
4. 调整 TensorWrapper：在 codec.py 中，将 if tensor.is_cuda: 扩展为 if tensor.is_cuda or tensor.is_npu:，确保 NPU tensor 也能触发 CPU 回退。
5. 新增文档：创建 docs_new/docs/hardware-platforms/ascend-npus/diffusion/disaggregation.mdx，介绍 NPU 上分解的使用方法，包括 Mooncake 安装配置。

关键文件：

- python/sglang/multimodal_gen/runtime/disaggregation/scheduler_mixin.py（模块 调度器；类别 source；类型 dependency-wiring；符号 _init_disagg_request_scheduler, _init_disagg_state, _init_disagg_transfer_manager, _prefetch_transfer_ready）：核心调度混合类，所有设备字符串和 Stream 创建均替换为平台无关 API，是本次变更的主入口。
- python/sglang/multimodal_gen/runtime/disaggregation/transport/manager.py（模块 传输层；类别 source；类型 dependency-wiring；符号 stage_tensors, stage_tensors_async, load_tensors_async, load_tensors）：传输管理器，替换了 Stream/Event 类型、同步调用和默认设备参数，是跨平台传输的关键。
- python/sglang/multimodal_gen/runtime/disaggregation/transport/buffer.py（模块 传输层；类别 source；类型 core-logic；符号 write_tensor, read_tensor, read_tensors_from_manifest, write_tensors_from_gpu）：传输缓冲区，所有 Stream 相关操作（创建、上下文管理器）替换为 get_device_module()，是跨平台副本的基础。
- python/sglang/multimodal_gen/runtime/disaggregation/transport/codec.py（模块 序列化；类别 source；类型 core-logic；符号 TensorWrapper.init）：TensorWrapper 需要支持 NPU tensor 回退 CPU，该修改确保 NPU tensor 不会被错误地留在设备上。
- docs_new/docs/hardware-platforms/ascend-npus/diffusion/disaggregation.mdx（模块 文档；类别 other；类型 documentation）：新增 NPU 平台扩散分解使用指南，包括 Mooncake 安装步骤。

关键符号：_init_disagg_request_scheduler, _init_disagg_state, _init_disagg_transfer_manager, _prefetch_transfer_ready, _broadcast_req_to_all_ranks, _disagg_prefetch_event_loop, stage_tensors, stage_tensors_async, load_tensors_async, load_tensors, write_tensor, read_tensor, read_tensors_from_manifest, write_tensors_from_gpu, TensorWrapper.init

关键源码片段

python/sglang/multimodal_gen/runtime/disaggregation/scheduler_mixin.py

核心调度混合类，所有设备字符串和 Stream 创建均替换为平台无关 API，是本次变更的主入口。

```
# scheduler_mixin.py 中设备字符串和 Stream 创建替换示例

def _init_disagg_request_scheduler(self: Scheduler, req: Req) -> None:
    scheduler_template = self.worker.pipeline.get_module("scheduler")
    if scheduler_template is None:
        return
    # 使用 current_platform.device_type 替换固定的 "cuda"
    device = torch.device(f"{current_platform.device_type}:{self.worker.local_rank}")
    _init_request_scheduler_from_template(scheduler_template, req, device)

def _init_disagg_state(self: Scheduler, server_args, local_rank: int) -> None:
    # ...
    # 替换 torch.cuda.Stream 为 torch.get_device_module().Stream
    device = torch.device(f"{current_platform.device_type}:{local_rank}")
    self._transfer_stream = torch.get_device_module().Stream(device=device)
```

python/sglang/multimodal_gen/runtime/disaggregation/transport/manager.py

传输管理器，替换了 Stream/Event 类型、同步调用和默认设备参数，是跨平台传输的关键。

```
# manager.py 中同步和事件创建替换示例

class DiffusionTransferManager:
    def stage_tensors(self, request_id, tensor_fields, scalar_fields=None,
                      stream: torch.Stream | None = None) -> StagedTransfer | None:
        # ... 分配和写入 ...
        if stream is not None:
            stream.synchronize()
        elif torch.get_device_module().is_available():
            torch.get_device_module().synchronize()
        # ...

    def load_tensors_async(self, request_id, manifest,
                           device: torch.device | str = current_platform.device_type,
                           stream: torch.Stream | None = None) -> tuple[dict, torch.Event | None]:
        # ... 加载逻辑，使用 stream 记录事件
        load_event = torch.get_device_module().Event()
        load_event.record(stream)
        return tensors, load_event
```

python/sglang/multimodal_gen/runtime/disaggregation/transport/buffer.py

传输缓冲区，所有 Stream 相关操作（创建、上下文管理器）替换为 get_device_module()，是跨平台副本的基础。

```
# buffer.py 中通用 stream 上下文替换示例

class TransferTensorBuffer:
    def write_tensor(self, handle: SlotHandle, name: str, tensor: torch.Tensor,
                     byte_offset: int = 0,
                     stream: torch.Stream | None = None) -> int:
        # ... 复制 tensor bytes ...
        if stream is not None:
            with torch.get_device_module().stream(stream):
                dst.copy_(src_bytes, non_blocking=True)
        else:
            dst.copy_(src_bytes, non_blocking=True)
        return nbytes

    def read_tensor(self, handle, shape, dtype, byte_offset=0, device="cpu",
                    stream: torch.Stream | None = None) -> torch.Tensor:
        # ... 读取 raw 并 reshape ...
        if stream is not None:
            with torch.get_device_module().stream(stream):
                return src.to(device, non_blocking=True)
```

```
    return src.to(device, non_blocking=True)
```

评论区精华

Review 中主要的讨论集中在以下方面：

- 类型注解安全问题：*gemini-code-assist[bot]* 指出多处使用 torch.get_device_module() 在类型注解中会导致运行时错误，建议改用 Any。作者统一替换为 torch.Stream 和 torch.Event，避免了该问题。
- 文档路径硬编码：*gemini-code-assist[bot]* 建议使用动态方式定位 mooncake 库路径，但作者认为当前路径可用，未修改；同时 ping1jing2 提出当前 NPU 暂不支持 Mooncake，该文档后续需要更新。
- NPU Mooncake 支持：ping1jing2 表示 NPU 尚未支持 Mooncake，作者回复文档中的安装指南本人验证可运行，但未明确确认正式支持状态。
  - 类型注解中使用函数调用导致运行时错误 (correctness): 作者将注解替换为 torch.Stream 和 torch.Event，避免了运行时错误。
  - 文档中 LD_LIBRARY_PATH 硬编码 (documentation): 作者认为当前路径可用，未修改；ping1jing2 指出 NPU 暂不支持 Mooncake，该文档后续需要更新。
  - NPU 平台是否支持 Mooncake (question): 作者回复 'this works for me' 并引用 Mooncake 文档，但未确认正式支持状态。

风险与影响

- 风险：
  1. 类型注解风险（已修复）：初始提交中使用 torch.get_device_module().Stream 作为类型注解，在 Python 中函数调用不可用于类型注解，可能导致运行时错误。已在最终版本修正为 torch.Stream 等合法类型。
  2. 文档路径硬编码：文档中 *export* LD_LIBRARY_PATH=... 包含具体 Python 版本路径，在其他环境可能不可用，增加用户配置难度。
  3. NPU Mooncake 支持不确定性：NPU 平台是否正式支持 Mooncake 未确认，文档可能引导用户尝试尚未完全兼容的功能。
  4. 回归风险：替换 torch.cuda.* 为 torch.get_device_module().* 在 CUDA 环境下应兼容，但若 get_device_module() 返回的模块接口有差异，可能引入回归。需要充分测试。
- 影响：
  - 用户影响：NPU 用户现在可以像 CUDA 一样使用扩散模型分解功能，提升推理灵活性和性能。CUDA 用户无影响。
  - 系统影响：核心传输层和调度器现在通过 current_platform 和 get_device_module() 抽象设备，后续增加新平台只需实现对应接口。
  - 团队维护：需确保跨平台测试覆盖，避免因抽象层引发性能退化。
  - 风险标记：类型注解风险（已修复），文档路径兼容性，NPU Mooncake 支持状态未知

关联脉络

- PR #25904 :memo: docs(diffusion): add MXFP4 quantization docs: 同为 NPU 扩散功能完善，且涉及 Ascend NPU 平台文档，表明 NPU 支持持续扩展。
- PR #26267 [NPU] Add torchaudio dependency for NPU platform: 同为 NPU 平台基础设施完善，与当前 PR 共同推动 NPU 平台可用性。