PR #36832 完整报告

sgl-project/sglang

[Diffusion] Avoid direct GPU parameter copies

合并时间：2026-08-29 22:43

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36832

执行摘要

- 一句话：Direct GPU 加载下兼容权重零拷贝成为参数存储
- 推荐动作：值得精读。重点关注 _can_assign_tensor_without_copy 的 fail-closed 门控设计、data_ptr 断言如何锁定两个加载路径的行为契约，以及本 PR 从“全量采用”到“仅显式 Direct GPU 加载采用”的演进过程——这是处理“优化作用域过宽导致回归”的典型范例。可与 #36902、#36931 对照阅读 diffusion 组件加载的演进脉络。

功能与动机

PR body 明确说明：让兼容的 TP=1 linear checkpoint 张量在 Direct GPU loading 期间成为参数存储，避免直接 GPU 副本，同时保留针对 TP sharding、量化、transformed mappings、特殊参数、layout 和 stride 的 fail-closed 门控。作者在 Issue 评论中补充：第一版实现把设备张量采用扩展到了普通 CUDA 加载，导致 Z-Image 显存从 14,196 MiB 上升到 14,406 MiB、LTX2 增加 512 MiB，因此后续提交将设备张量采用限定到显式 load_full_state_dict_on_device 的 Direct GPU 路径，普通加载必须保持既有物化路径。

实现拆解

1. 入口与参数传递：transformer_loader.py 的 load_customized 在 --direct-gpu-weight-loading 开启时通过 WeightLoadPlan.for_component(..., load_full_state_dict_on_device=direct_gpu_weight_loading) 标记 Direct GPU 加载，maybe_load_fsdp_model 再将它映射为 load_model_from_full_model_state_dict 的新参数 allow_device_tensor_assignment（默认 False）。
2. 零拷贝判定逻辑：fsdp_load.py 将 _can_assign_cpu_tensor_without_copy 泛化为 _can_assign_tensor_without_copy，去掉 device 限制，并新增 full_tensor.layout == target_param.layout 和 stride() 一致性检查；保留原有 fail-closed 门控：仅 TP=1、未量化线性层、普通 nn.Parameter、无 is_metadata/is_sharded_weight/needs_scalar_to_array 标记时允许直接采用。
3. 赋值决策：在 load_model_from_full_model_state_dict 中，仅当 full_tensor 在 CPU 上（保持既有 CPU checkpoint 采用路径）或 allow_device_tensor_assignment=True 且通过 _can_assign_tensor_without_copy 检查时，sharded_tensor = full_tensor；其余情况维持原物化复制逻辑。这样普通 CUDA 加载不会意外采用 checkpoint 存储。
4. 测试配套：test_fsdp_load.py 新增 _load_replicated_weight 辅助方法，用 data_ptr 断言锁定四个行为契约：CPU checkpoint 存储采用、普通 CUDA 加载保持物化路径、Direct

CUDA 加载采用 checkpoint 存储、不兼容 layout 或 TP 权重的零拷贝赋值被拒绝；另外将 AMD ROCm 的 FP8 dtype 断言调整为接受原生 float8_e4m3fnuz 表示。

5. 文档与提示：docs/docs/sglang-diffusion/api/cli.mdx 更新 --direct-gpu-weight-loading 的参数说明，transformer_loader.py 的启动 warning 改为说明兼容张量成为模型存储、变换张量仍可能需要临时 GPU 分配。

关键文件：

- python/sglang/multimodal_gen/runtime/loader/fsdp_load.py（模块 加载器；类别 source；类型 core-logic；符号 _can_assign_tensor_without_copy, load_model_from_full_model_state_dict, maybe_load_fsdp_model）：核心逻辑所在：将零拷贝判定从 CPU-only 泛化为设备无关的 _can_assign_tensor_without_copy，新增 allow_device_tensor_assignment 开关，决定 checkpoint 张量何时直接成为参数存储而不复制。
- python/sglang/multimodal_gen/test/unit/test_fsdp_load.py（模块 加载器；类别 test；类型 test-coverage；符号 _load_replicated_weight, test_tp1_unquantized_linear_adopts_cpu_checkpoint_storage, test_ordinary_cuda_loading_preserves_materialization_path, test_direct_cuda_loading_adopts_checkpoint_storage）：行为契约测试：用 data_ptr 断言锁定 CPU 采用、普通 CUDA 物化、Direct CUDA 采用和不兼容拒绝四条路径，防止零拷贝作用域再次扩大导致显存回归。
- python/sglang/multimodal_gen/runtime/loader/component_loaders/transformer_loader.py（模块 加载器；类别 source；类型 core-logic）：Direct GPU 加载入口处更新启动 warning 文案，告知兼容张量会成为模型存储、变换张量仍可能临时分配，帮助用户理解显存行为。
- docs/docs/sglang-diffusion/api/cli.mdx（模块 文档；类别 docs；类型 documentation）：同步 --direct-gpu-weight-loading 参数文档，说明兼容张量零拷贝成为存储、变换张量仍可能临时分配。

关键符号：_can_assign_tensor_without_copy, load_model_from_full_model_state_dict, maybe_load_fsdp_model, _load_replicated_weight

关键源码片段

python/sglang/multimodal_gen/runtime/loader/fsdp_load.py

核心逻辑所在：将零拷贝判定从 CPU-only 泛化为设备无关的 _can_assign_tensor_without_copy，新增 allow_device_tensor_assignment 开关，决定 checkpoint 张量何时直接成为参数存储而不复制。

```
def _can_assign_tensor_without_copy(
    actual_param: torch.nn.Parameter,
    full_tensor: torch.Tensor,
    target_param: torch.Tensor,
) -> bool:
    """判定 TP=1 未量化线性层是否可以直接采用 checkpoint 张量作为参数存储。
```

```
    这是 fail-closed 门控：任何不满足条件的场景都回到复制物化路径，
    避免 loader 在需要变换或切分的权重上意外采用 checkpoint 存储。
    """
    weight_loader = actual_param.__dict__.get("weight_loader")
    if not isinstance(weight_loader, MethodType):
        return False # 没有 weight_loader 的参数无法保证赋值语义一致

    owner = weight_loader.__self__
    if not isinstance(
        owner,
        (ReplicatedLinear, ColumnParallelLinear, RowParallelLinear),
    ):
        return False # 只对线性层开放零拷贝采用
    if not isinstance(owner.quant_method, UnquantizedLinearMethod):
        return False # 量化权重布局可能被改写，禁止零拷贝
    if not isinstance(owner, ReplicatedLinear) and owner.tp_size != 1:
        return False # TP 切分会产生分片视图，不能直接采用原始 checkpoint 张量
    if type(actual_param) is not nn.Parameter:
        return False # 特殊参数子类可能携带额外语义
    if any(
        actual_param.__dict__.get(attribute, False)
        for attribute in (
            "is_metadata",
            "is_sharded_weight",
            "needs_scalar_to_array",
        )
    ):
        return False # 需要变换的权重必须走复制物化
    return (
        full_tensor.shape == target_param.shape
        and full_tensor.dtype == target_param.dtype
        and full_tensor.layout == target_param.layout
        and full_tensor.stride() == target_param.stride()
    )

# maybe_load_fsdp_model 只在显式 Direct GPU 加载时打开设备张量采用
load_model_from_full_model_state_dict(
    model,
    weight_iterator,
    weight_load_plan.checkpoint_load_device,
    param_dtype,
    strict=strict,
    cpu_offload=load_on_cpu,
    param_names_mapping=param_names_mapping_fn,
    keep_checkpoint_mapping=keep_checkpoint_mapping,
    allow_device_tensor_assignment=(
        weight_load_plan.load_full_state_dict_on_device
    ),
    preconverted_state_dict=preconverted_state_dict,
```

```
)

# 加载器内部的关键赋值决策
elif weight_loader is not None:
    assert actual_param is not None
    if (
        full_tensor.device.type == "cpu" or allow_device_tensor_assignment
    ) and _can_assign_tensor_without_copy(
        actual_param, full_tensor, meta_sharded_param
    ):
        # checkpoint 张量本身兼容且落在允许的设备上，直接采用它作为参数存储
        sharded_tensor = full_tensor
    else:
        # 其余情况保持既有的物化复制路径，避免意外采用 checkpoint 存储
        ...
```

python/sglang/multimodal_gen/test/unit/test_fsdp_load.py

行为契约测试：用 data_ptr 断言锁定 CPU 采用、普通 CUDA 物化、Direct CUDA 采用和不兼容拒绝四条路径，防止零拷贝作用域再次扩大导致显存回归。

```
class TestOrdinaryWeightLoading(unittest.TestCase):
    def _load_replicated_weight(
        self, device: torch.device, *, allow_device_tensor_assignment: bool = False
    ) -> tuple[torch.nn.Parameter, torch.Tensor]:
        # 在 meta 设备上搭好模型，然后把一个 4x4 float32 张量当作
        # checkpoint 权重交给 FSDP 加载器。
        with torch.device("meta"):
            model = _ReplicatedLinearModel()
        checkpoint_weight = torch.arange(
            16, dtype=torch.float32, device=device
        ).reshape(4, 4)

        fsdp_load.load_model_from_full_model_state_dict(
            model,
            iter((("proj.weight", checkpoint_weight),)),
            checkpoint_load_device=device,
            param_dtype=torch.float32,
            strict=True,
            param_names_mapping=fsdp_load.get_param_names_mapping({}),
            allow_device_tensor_assignment=allow_device_tensor_assignment,
        )
        return model.proj.weight, checkpoint_weight

    @unittest.skipUnless(torch.cuda.is_available(), "CUDA is required")
    def test_ordinary_cuda_loading_preserves_materialization_path(self):
        # 普通 CUDA 加载必须保持复制物化，避免意外采用 checkpoint 存储
        model_weight, checkpoint_weight = self._load_replicated_weight(
            torch.device("cuda:0")
        )
```

```
        self.assertNotEqual(model_weight.data_ptr(), checkpoint_weight.data_ptr())
        torch.testing.assert_close(model_weight, checkpoint_weight)

    @unittest.skipUnless(torch.cuda.is_available(), "CUDA is required")
    def test_direct_cuda_loading_adopts_checkpoint_storage(self):
        # 显式 Direct GPU 加载才允许 checkpoint 张量直接成为参数存储
        model_weight, checkpoint_weight = self._load_replicated_weight(
            torch.device("cuda:0"), allow_device_tensor_assignment=True
        )
        self.assertEqual(model_weight.data_ptr(), checkpoint_weight.data_ptr())
        torch.testing.assert_close(model_weight, checkpoint_weight)
```

评论区精华

作者在 Issue 评论中披露了三个关键决策，本 PR 无外部 review 评论：

- 零拷贝作用域必须收窄：第一版让普通 CUDA 加载也采用 checkpoint 存储，造成 Z-Image（+210 MiB）和 LTX2（+512 MiB）显存回归；提交 30a29940 将设备张量采用限定到显式 Direct GPU 加载，普通加载保持复制物化。这是本 PR 最重要的设计修正。
- AMD FP8 dtype 是平台契约差异：MI300 原生产生 float8_e4m3fnuz，而测试断言的是 CUDA 的 float8_e4m3fn；作者先开了 #36930，随后直接折叠进本分支（ade56968），避免与加载器改动耦合。
- AMD CI 的 EAGLE 失败与 Direct GPU 无关：extra-a-test-1-gpu-small-amd 在 AITER 执行后出现 GPU 内存访问错误，Direct GPU 路径 CUDA-only 且 AMD 上未选中，判定为独立 CI 问题。
  - 零拷贝采用的作用域必须限定到显式 Direct GPU 加载 (design): 设备张量采用仅在显式 Direct GPU 加载时启用，普通 CUDA 加载保持复制物化路径。
  - AMD ROCm 上 FP8 dtype 的平台差异 (testing): 测试断言改为接受 AMD 原生 FP8 dtype 表示，与 Direct GPU 加载行为无关。
  - AMD CI 中 EAGLE 测试失败与 Direct GPU 路径无关 (other): 判定为独立 CI 问题，不需要本 PR 修改源码。

风险与影响

- 风险：
  - 零拷贝存储生命周期：采用 checkpoint 张量后，模型参数与 checkpoint 共享存储；若后续有 in-place 修改参数或释放 checkpoint 引用的代码，可能引入数据一致性或生命周期纠缠，目前测试只覆盖单个线性层。
  - 核心加载路径变更：load_model_from_full_model_state_dict 是 FSDP 加载主路径，新增 allow_device_tensor_assignment 分支后，两个加载路径（普通 / Direct）行为不同，后续维护需警惕回归。
  - 显存尖峰未完全消除：Direct GPU 加载下仍需要变换、量化、TP 切分的权重会继续产生临时 GPU 分配，峰值显存改善幅度模型相关。

- AMD 平台差异：FP8 dtype 断言改动是测试层面的平台适配，需要防止它掩盖未来真实的精度回归。
- 影响：影响范围集中在 diffusion 模型的 Direct GPU weight loading 启动路径：启用 --direct-gpu-weight-loading 的用户可减少一次 GPU 参数复制，降低启动峰值显存和耗时；普通加载路径行为完全不变（有 data_ptr 测试保证）。对团队而言，加载器新增了行为分支和门控条件，测试维护成本略增，但文档和 warning 已同步，边界清晰。
- 风险标记：核心加载路径变更，零拷贝存储生命周期，显存回归已修复，AMD 平台差异测试调整

关联脉络

- PR #36931 [diffusion] Honor explicit component offload: 同为 diffusion 加载配置的显式控制与冲突拒绝，与 Direct GPU 加载的 fail-closed 门控是同一设计线。
- PR #36902 [Diffusion] Delegate recognized quantized components to Transformers: 同在 component loader 边界处理量化组件，本 PR 的零拷贝门控也把量化权重排除在外。
- PR #36863 [diffusion] Fix image encoder parallel folding proposal: 同属 diffusion 组件加载配置修复，验证同一 loader 流程的行为。
- PR #36905 [Diffusion] Honor explicit offload in resident requirements: diffusion 显式 offload/ 驻留规则校验，与 Direct GPU 加载的配置限制相互补充。