PR #35618 完整报告

sgl-project/sglang

[diffusion] UX: report where a component's weights are

合并时间：2026-08-20 13:05

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35618

执行摘要

- 一句话：加载日志按内存类型报告组件权重驻留位置
- 推荐动作：值得精读。虽然只是日志增强，但 component_residency_bytes() 的实现有几个可复用的设计：用 /proc/self/maps 区分 file-backed 与 anonymous host 内存、用 storage 指针去重避免扁平 buffer 重复计数、pinned 判断先于 file-backed 判断（因为 CUDA host allocator 背后是命名映射）。对任何需要诊断显存 / 内存占用的模块都有参考价值。

功能与动机

PR body 指出：加载器报告的 consumed GPU mem 是加载前后空闲显存的差值，对任何 offloaded 组件该差值都是 0.00 GB，日志读起来像组件零成本（model size: 21.16 GB, consumed GPU mem: 0.00 GB）。而 21.16 GB went somewhere，运行期没有任何信息说明权重真正在哪里，导致在显存受限的卡上无法区分驻留权重与流式权重，也无法区分 pinned host 内存（内核不可回收）与普通 host 内存。

实现拆解

1. 新增地址空间归类工具（python/sglang/multimodal_gen/runtime/loader/utils.py）：新增 _read_process_mappings() 读取 /proc/self/maps，按地址排序返回 (start, end, is_file_backed) 三元组；新增 component_residency_bytes() 遍历模块的 parameters()、buffers() 以及 layerwise_offload_managers 的 iter_cpu_weights()，以 untyped_storage().data_ptr() 去重并归入 vram、host_pinned、host_mapped、host 四个桶；新增 format_component_residency() 仅渲染非空桶，host 显示为 host pageable，host_mapped 显示为 host mapped。
2. 改造组件加载日志行（python/sglang/multimodal_gen/runtime/loader/component_loaders/component_loader.py）：load() 中的 logger.info 移除 consumed GPU mem: %.2f GB，改用 format_component_residency(component) 输出 resident in ... 形式，同时保留 avail GPU mem。这样加载时点（layerwise offload 配置之前约 20 s）的驻留位置能直接可见。
3. 改造 layerwise offload 启用日志（python/sglang/multimodal_gen/runtime/managers/memory_managers/layerwise_offload.py）：configure_layerwise_offload_modules() 中原本只输出组件名列表，现在逐个组件附上 format_component_residency(modules[name])，报告卸载后的最终放置（例如 vram 4.63 GB, host pinned 17.25 GB），弥补加载行只

能描述预卸载放置的缺口。

4. 新增单元测试（python/sglang/multimodal_gen/test/unit/test_component_residency_report.py，10 个用例）：覆盖 host 权重计数、同一扁平 buffer 的多个切片只计一次、空占位符跳过、file-backed 与 anonymous 分离、pinned 优先于 file-backed、resident 参数入 vram、pinned host 分离、非 nn.Module 返回空、格式化只列非零桶、无权重组件输出 weights: none；其中 device/pinned 用例在无 CUDA 时跳过，mapping 用例在无 /proc 时跳过。
5. 配套说明：无配置、部署或 schema 变更；改动全部集中在启动期日志路径，不触碰 serving 热路径。

关键文件：

- python/sglang/multimodal_gen/runtime/loader/utils.py（模块 加载工具；类别 source；类型 core-logic；符号 _read_process_mappings, component_residency_bytes, is_file_backed, add）：承载核心实现：新增 _read_process_mappings()、component_residency_bytes()、format_component_residency()，负责把组件权重按内存类型分桶并渲染日志。
- python/sglang/multimodal_gen/test/unit/test_component_residency_report.py（模块 单元测试；类别 test；类型 test-coverage；符号 _FakeOffloadManager, init, iter_cpu_weights, _Streamed）：新增 10 个单元测试，覆盖分桶、storage 去重、占位符跳过、file-backed 与 anonymous 分离、pinned 优先及格式化规则，是本次变更正确性的主要保障。
- python/sglang/multimodal_gen/runtime/managers/memory_managers/layerwise_offload.py（模块 逐层卸载；类别 source；类型 dependency-wiring）：layerwise offload 启用日志从只列组件名升级为逐个组件报告最终驻留位置，弥补加载行只能描述预卸载放置的缺口。
- python/sglang/multimodal_gen/runtime/loader/component_loaders/component_loader.py（模块 组件加载；类别 source；类型 core-logic）：加载完成日志的消费入口：移除误导性的 consumed GPU mem，改为输出权重驻留位置。

关键符号：_read_process_mappings, component_residency_bytes, is_file_backed, add, format_component_residency

关键源码片段

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

新增 10 个单元测试，覆盖分桶、storage 去重、占位符跳过、file-backed 与 anonymous 分离、pinned 优先及格式化规则，是本次变更正确性的主要保障。

```
# python/sglang/multimodal_gen/test/unit/test_component_residency_report.py

class _FakeOffloadManager:
    """Stands in for LayerwiseOffloadManager's host-side weight store."""

    def __init__(self, tensors):
```

```
        self._tensors = tensors

    def iter_cpu_weights(self):
        for index, tensor in enumerate(self._tensors):
            yield f"w{index}", tensor


class _Streamed(nn.Module):
    """A module whose real weights live in its managers, not its parameters."""

    def __init__(self, managers):
        super().__init__()
        # layerwise offload leaves (1,) placeholders behind
        self.placeholder = nn.Parameter(torch.empty(0), requires_grad=False)
        self.layerwise_offload_managers = managers


class TestComponentResidencyBytes:
    def test_slices_of_one_buffer_are_counted_once(self):
        # this is the layerwise layout: one flat buffer, many logical weights
        buffer = torch.empty(1024, dtype=torch.float32)
        views = [buffer[0:256], buffer[256:512], buffer[512:1024]]
        module = _Streamed([_FakeOffloadManager(views)])
        assert component_residency_bytes(module)["host"] == 4096

    def test_pinned_wins_over_the_file_backed_check(self):
        if not torch.cuda.is_available():
            pytest.skip("pinning needs CUDA")
        # CUDA's host allocator sits behind a named mapping, so the
        # file-backed check alone would call this one mapped
        pinned = torch.empty(1024, dtype=torch.float32, pin_memory=True)
        module = _Streamed([_FakeOffloadManager([pinned])])
        totals = component_residency_bytes(module)
        assert totals["host_pinned"] == 4096
        assert totals["host_mapped"] == 0
```

python/sglang/multimodal_gen/runtime/loader/component_loaders/component_loader.py

加载完成日志的消费入口：移除误导性的 consumed GPU mem，改为输出权重驻留位置。

```
# python/sglang/multimodal_gen/runtime/loader/component_loaders/component_loader.py
# 加载完成的日志行：不再输出“消耗了多少显存”，而是报告权重实际驻留位置。
logger.info(
    f"Loaded %s: %s ({source} version). model size: %s GB, %s. avail GPU mem: %.2f GB",
    component_name,
    component.__class__.__name__,
    model_size,
    format_component_residency(component), # 例如 "host pageable: 21.16 GB"
    current_gpu_mem,
```

）

评论区精华

本 PR 没有收到任何 review 评论；两条 issue 评论均为作者发出的 /tag-and-rerun-ci（对应 CI Extra 曾失败后重跑）。作者在 PR body 中自行阐述了两个必须算对的会计细节：一是 layerwise-offloaded 权重不在 parameters() / buffers() 中（模块只留 (1,) 占位符），必须额外遍历 offload 管理器的 iter_cpu_weights()；二是大小必须取自 storage 并按 storage 去重，因为一块扁平 host buffer 支撑多个逻辑权重，逐 tensor 求和会重复计数。此外还说明了 host_mapped 统计的是映射大小而非驻留页，真实占用至多等于报告值。

- CI Extra 失败与重跑 (other): 无 review 评论；变更由作者自查并附带完整精度与速度测试说明。

风险与影响

- 风险：
  1. 平台相关：/proc/self/maps 只存在于 Linux，非 Linux 下 _read_process_mappings() 返回 None，host 字节不再拆分 file-backed 与 anonymous，但仍能报告总量，功能降级而非失效。
  2. 计数口径：host_mapped 报告的是文件映射区间大小而非实际驻留页，映射的 safetensors 文件按需缺页加载时可能虚高，日志读者需要知道这是上界。
  3. 日志时机：加载行在 layerwise offload 配置前约 20 s 运行，只能描述预卸载放置；若用户只看第一行可能误判，必须结合 offload 启用行（现已同步输出 settled placement）。
  4. 稳定性：untyped_storage() / is_pinned() 均有异常捕获保护，非 nn.Module 直接返回空字典，不会让加载流程崩溃。
  5. 性能：每个组件启动期多跑两次 /proc 解析与指针二分查找，不在 serving 路径，实测 per-step 时间不变（1.03 s）。 - 影响：对用户（部署者）：显存受限场景下日志首次能回答 " 权重到底在哪 "，可区分驻留权重与流式权重、pinned 与 pageable，有助于判断是否该开 layerwise offload 或扩容。对系统：无运行时行为变化，仅日志文案变化，同一运行的输出 SHA-256 (78a2a963…) 与改动前一致。对团队：diffusion 子系统的可观测性提升，为后续 offload 策略调优提供诊断依据；测试覆盖了分桶、去重、平台降级等关键分支，但对 CUDA 与 /proc 的依赖意味着部分用例在无对应环境的 CI 上会自动跳过。 - 风险标记：平台相关：依赖 /proc/self/maps，host_mapped 统计映射大小可能虚高，日志只描述加载时点，offload 后需看第二行日志，测试依赖 CUDA 与 /proc 环境会跳过

关联脉络

- PR #35615 [diffusion] ci: use canonical residency selector: 同属 diffusion 领域并围绕“驻留（residency）”概念，该 PR 统一 CI 对比中的驻留选择器，与本 PR 的驻留位置报告在概念上相互呼应。

- PR #35538 [diffusion] fix: stop reserving NCCL device buffers for single-rank groups: 同为 diffusion 显存占用治理，关注权重 / 缓冲区实际占用的可视化与优化，本 PR 为其提供日志侧的可观测性支撑。
- PR #35614 [diffusion] chore: reduce per-request log noise: 同为 diffusion 运行日志的可观测性改进方向，本 PR 是启动期日志的信息增强，两者共同提升 diffusion 部署的可诊断性。