PR #30157 完整报告

sgl-project/sglang

Size KV pool after CUDA graph capture (opt-in)

合并时间：2026-07-08 03:05

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30157

执行摘要

- 一句话：CUDA 图捕获后动态调整 KV 缓存池大小，减少内存浪费
- 推荐动作：推荐精读本 PR，涉及底层 CUDA VMM 实现和 KV 池动态调整的设计权衡。对于大规模推理服务，可评估启用以提升内存效率。注意当前 opt-in 状态和 HND 槽计算争议，建议在非默认路径下充分验证后再默认开启。

功能与动机

在 --mem-fraction-static 启发式中，KV 池大小在 CUDA 图捕获前确定，图形内存的占用是猜测值，在大配置下偏差可达 ~10 GB/GPU，导致部分配置浪费内存，部分配置面临 OOM。PR body 明确指出：'the KV pool is sized before CUDA-graph capture, so graph memory is a heuristic guess baked into mem_fraction_static. On large configs the guess can be off by ~10 GB/GPU; the padded activation reserve silently absorbs the difference, wasting memory on some configs and risking OOM on others.' 因此需要一种机制，在捕获图后根据实际空闲内存精确确定 KV 池大小。

实现拆解

1. 新增 CUDA 虚拟内存管理 arena (kv_vmm_backing.py)：创建 KvVmmArena 类，通过 CUDA VMM 接口预留大块虚拟地址空间（默认 256 GiB），不分配物理内存。提供 bump 分配器（内联 C++ 桩），保证指针按 granularity 对齐，支持 torch.cuda.CUDAPluggableAllocator。每个实例加载独立 .so，避免多进程符号冲突。
2. 修改 mem_fraction 计算路径 (server_args.py)：新增 post_capture_kv_sizing_planned() 方法，当环境变量启用且满足条件（CUDA、非 MLA、非 memory_saver 等）时，计算 mem_fraction_static 跳过图占用和激活预留，仅保留基础元数据和并行开销。
3. KV 池关联 VMM arena (memory_pool.py)：在 KVCache 及其子类中新增 post_capture_active 标志和 _post_capture_owner 引用。启用时 _create_buffers 使用 KvVmmArena 创建张量（仅虚拟地址）。新增 KvBufferDesc 辅助类精确描述缓冲区字节跨度，支持动态计算备份大小。
4. 新增图捕获后调整池大小的方法 (model_runner_kv_cache_mixin.py)：在 ModelRunnerKVCacheMixin 中添加 post_capture_resize_kv_pool()，在 CUDA 图捕获完成后调用。测量当前空闲 GPU 内存，减去必要的 headroom，计算 KV 池可用最大 token 数，调用 _finalize_backing 物理备份 VMA 范围。同时修改 _profile_available_bytes，在

启用时正确处理 mamba 预捕获预留。

5. 配套修改分配器和 SWA 池：更新 allocator/base.py 和 allocator/swa.py 添加 resize 方法；swa_memory_pool.py 添加 finalize_backing 协调子池备份。
6. 端到端测试 (test_post_capture_kv_sizing.py)：通过环境变量启动服务器，断言日志含有 'Post-capture KV sizing'、API 返回 max_total_num_tokens>0、GSM8K 准确率≥0.80。注册到 base-b CI。

关键文件：

- python/sglang/srt/mem_cache/kv_vmm_backing.py（模块 VMM 层；类别 source；类型 dependency-wiring；符号 _driver, _check, align_up, query_granularity）：新增文件，实现 CUDA 虚拟内存管理 arena，是 post-capture sizing 的核心基础。
- python/sglang/srt/model_executor/model_runner_kv_cache_mixin.py（模块 KV 编排；类别 source；类型 data-contract；符号 post_capture_kv_active, post_capture_resize_kv_pool, _resolve_memory_pool_config, _config_from_budget）：核心修改文件，添加 post_capture_resize_kv_pool 和 post_capture_kv_active 属性，是后调整的编排中心。
- python/sglang/srt/mem_cache/memory_pool.py（模块 内存池；类别 source；类型 core-logic；符号 KvBufferDesc, init, _rows, reserved_span_bytes）：修改了 KVCache 基类和子类，添加 post_capture_active 标志和 VMM 备份支持，新增 KvBufferDesc 辅助类。
- python/sglang/srt/server_args.py（模块 配置；类别 source；类型 configuration；符号 post_capture_kv_sizing_planned, mamba_pre_capture_reserve_mb, reserve_for_graph_mb, reserve_for_deepep_a2a_mb）：修改 mem_fraction 计算路径，新增 post_capture_kv_sizing_planned 方法决定何时跳过图预留。
- test/registered/mem_cache/test_post_capture_kv_sizing.py（模块 测试；类别 test；类型 test-coverage；符号 TestPostCaptureKVSizing, setUpClass, tearDownClass, _server_logs）：新增端到端测试，验证 post-capture sizing 路径执行、API 响应和 GSM8K 准确率。

关键符号：KvVmmArena.init, post_capture_resize_kv_pool, _profile_available_bytes, post_capture_kv_sizing_planned, KvBufferDesc, finalize_backing, KVCache._create_buffers, SWAKVPool.finalize_backing

关键源码片段

python/sglang/srt/mem_cache/kv_vmm_backing.py

新增文件，实现 CUDA 虚拟内存管理 arena，是 post-capture sizing 的核心基础。

```
# 文件 python/sglang/srt/mem_cache/kv_vmm_backing.py ( 新增 )

from __future__ import annotations
import ctypes, logging, os, tempfile
from math import prod
from typing import TYPE_CHECKING, List, Optional, Sequence
import torch, torch.utils.cpp_extension
```

```
from torch.cuda.memory import CUDAPluggableAllocator

logger = logging.getLogger(__name__)
_drv = None

def _driver():
    """延迟加载 CUDA driver bindings."""
    global _drv
    if _drv is None:
        from cuda.bindings import driver
        _drv = driver
    return _drv

def _check(result, label: str):
    """包装 CUDA API 调用，失败时抛出 RuntimeError."""
    drv = _driver()
    err = result[0] if isinstance(result, tuple) else result
    if err != drv.CUresult.CUDA_SUCCESS:
        raise RuntimeError(f"{label} failed: {err}")
    return result[1] if isinstance(result, tuple) and len(result) > 1 else None

def _stub_source(sfx: str) -> str:
    """
    生成 C++ 桩代码，实现线程安全的 bump 分配器。
    符号按后缀区分，每个 arena 实例加载独立 .so，避免多进程/多实例冲突。
    """
    return f"""
#include <cstddef>
#include <cstdint>
#include <mutex>
extern "C" {{
static uintptr_t g_base = 0;
static size_t g_cursor = 0;
static size_t g_reserved = 0;
static size_t g_align = 512;
static std::mutex g_mu;
void kvarena_set_base_{sfx}(uintptr_t b) {{ g_base=b; g_cursor=0; }}
void kvarena_set_reserved_{sfx}(size_t r) {{ g_reserved=r; }}
void* kvarena_malloc_{sfx}(size_t size, int device, void* stream) {{
    size_t need = g_cursor + align_up(size, g_align);
    if (need > g_reserved) return 0;
    void* p = reinterpret_cast<void*>(g_base + g_cursor);
    g_cursor = need;
    return p;
}}
void kvarena_free_{sfx}(void* ptr, size_t size, int device, void* stream) {{}}
}}
"""
```

```
class KvVmmArena:
    """单个设备上的 CUDA 虚拟内存预留，对外暴露为 torch.cuda.MemPool."""
    _instance_count = 0

    def __init__(self, device_id: int, reserve_bytes: int = 256 * 1024**3):
        self.device_id = int(device_id)
        # 使用 PID 避免多进程编译 .so 冲突
        self._sfx = f"{KvVmmArena._instance_count}_{os.getpid()}"
        KvVmmArena._instance_count += 1
        drv = _driver()
        with torch.cuda.device(self.device_id):
            _check(drv.cuInit(0), "cuInit")
            self._prop = drv.CUmemAllocationProp()
            self._prop.type = drv.CUmemAllocationType.CU_MEM_ALLOCATION_TYPE_PINNED
            self._prop.location.type = drv.CUmemLocationType.CU_MEM_LOCATION_TYPE_DEVICE
            self._prop.location.id = self.device_id
            self.granularity = query_granularity(self.device_id)
            self._access = drv.CUmemAccessDesc()
            self._access.location.type = drv.CUmemLocationType.CU_MEM_LOCATION_TYPE_
            DEVICE
            self._access.location.id = self.device_id
            self._access.flags = drv.CUmemAccess_flags.CU_MEM_ACCESS_FLAGS_PROT_
            READWRITE
            self.reserved = self._align(reserve_bytes)
            # 预留虚拟地址空间
            self.base = int(
                _check(
                    drv.cuMemAddressReserve(self.reserved, self.granularity, 0, 0),
                    "cuMemAddressReserve"
                )
            )
            # 编译 C++ 桩并设置基址和上限（省略具体编译步骤）
```

python/sglang/srt/model_executor/model_runner_kv_cache_mixin.py

核心修改文件，添加 post_capture_resize_kv_pool 和 post_capture_kv_active 属性，是后调整的编排中心。

```
# 文件 python/sglang/srt/model_executor/model_runner_kv_cache_mixin.py（修改）

class ModelRunnerKVCacheMixin:

    @property
    def post_capture_kv_active(self: ModelRunner) -> bool:
        """判断当前是否启用 post-capture KV sizing (在 draft 上禁用)."""
        return (
            self.server_args.post_capture_kv_sizing_planned()
            and current_platform.is_cuda()
            and not self.is_draft_worker
        )
```

```
def post_capture_resize_kv_pool(self: ModelRunner) -> None:
    """
    在 CUDA 图捕获完成后调用，根据实际空闲内存重新确定 KV 池大小并物理备份。
    """
    pool = self.token_to_kv_pool
    torch.cuda.synchronize()
    free_gb = get_available_gpu_memory(
        self.device, self.gpu_id,
        distributed=get_world_group().world_size > 1,
        cpu_group=get_world_group().cpu_group,
    )
    headroom_gb = self.pre_model_load_memory * (1 - self.mem_fraction_static)
    decode_cfg = self.server_args.cuda_graph_config.decode
    eager_gap = (
        self.server_args.disaggregation_mode != "prefill"
        and decode_cfg.backend != Backend.DISABLED
        and decode_cfg.max_bs < self.max_running_requests
    )
    if eager_gap or self.mambaish_config is not None:
        headroom_gb = max(
            headroom_gb,
            self.server_args.mamba_pre_capture_reserve_mb(
                get_device_memory_capacity(self.device)
            ) / 1024,
        )
    pool_gb = max(free_gb - headroom_gb, 0)
    # 将可用 GB 转换为 token 数（具体计算略）
    max_total_num_tokens = int(pool_gb * 1024**3 / self.pool_token_size_bytes)
    logger.info("Post-capture KV sizing: max_total_num_tokens=%d", max_total_num_tokens)
    if hasattr(pool, "finalize_backing"):
        pool.finalize_backing(max_total_num_tokens)
    self.dynamic_max_total_num_tokens = max_total_num_tokens
```

评论区精华

- gemini-code-assist[bot] 的高优先级问题：
  - cuMemAddressReserve 未包 _check → 已修复。
  - 多进程库名竞态 → 已添加 PID 后缀。
  - 单位混淆（字节 vs MB） → 作者澄清无问题。
  - HND 分页槽计算错误 → 作者反驳，未完全解决。
  - 释放顺序不安全 → 已调整。
- merrymercy 的设计评论：
  - fallback headroom 过于宽松 → 后续 commit 优化为只在必要场景保留。
  - 不要用 getattr → 已移除 getattr 改用显式属性。
- 未解决争议：HND 槽计算逻辑仍需进一步验证。

- cuMemAddressReserve 缺少错误检查 (correctness): 已在后续 commit 中修复，添加 _check 包装。
- 多进程环境下库名竞态 (correctness): 已添加 os.getpid() 到后缀，保证唯一性。
- get_device_memory_capacity 单位混淆 (correctness): 无实际操作问题，但文档需澄清。
- HND 分页 KV 缓存槽计算错误 (correctness): 未完全达成一致，但代码保持原样。需要进一步验证。
- 释放顺序不安全 (correctness): 已调整删除顺序，先删张量再释放 owner。
- fallback headroom 过于宽松 (design): 后续 commit 优化了条件，仅在 eager gap 或 mamba 场景保留 headroom。
- 不要使用 getattr 访问 post_capture_active (style): 已移除 getattr，所有子类显式声明 post_capture_active 属性。

风险与影响

- 风险:
  - CUDA VMM 兼容性：依赖较新驱动和 GPU，在旧硬件或非 NVIDIA 平台上可能启动失败（已有条件检查）。
  - 多进程竞态：虽已加入 PID 后缀，但多个进程共享临时目录时仍可能冲突（低概率）。
  - 激活 headroom 逻辑：headroom 计算依赖 mem_fraction_static 准确性，若估算不足可能导致 OOM；已有 eager decode 和 mamba 保护，但未覆盖全部边缘。
  - HND 槽计算争议：若 review 中问题未被正确修复，可能导致物理内存过度分配或不足，影响大量用户。
  - 测试覆盖不足：仅测试了 Llama-3.1-8B 单模型，未覆盖 MLA、DeepSeek V4、MoE 等架构，也未测试多 GPU 场景。
  - CI 负担：测试注册在 base-b，每 PR 执行约 4 分钟，可能增加 CI 排队时间。
- 影响:
  - 用户：opt-in 特性，默认关闭，对现有用户无影响。启用用户可在大 GPU 上节省约 10 GB 内存，或避免 OOM。需了解 CUDA VMM 兼容性。
  - 系统：修改 KV 池分配核心路径和 mem_fraction 计算。启用时 KV 池初始化延迟到图捕获后，物理备份在后期单次完成，可能短暂增大抖动。
  - 团队：新增 424 行核心代码和测试，需要熟悉 CUDA VMM 的开发者维护。讨论成果提高了代码质量。
  - 风险标记：CUDA VMM 兼容性，多进程竞态，HND 槽计算争议，激活 headroom 逻辑，测试覆盖有限

关联脉络

- 暂无明显关联 PR