PR #43689 完整报告

vllm-project/vllm

[SharedOffloadRegion] Align blocks to page-size

合并时间：2026-06-03 19:25

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/43689

执行摘要

- 一句话：将 SharedOffloadRegion 块对齐到页大小以支持 O_DIRECT
- 推荐动作：建议阅读：该 PR 展示了如何通过类变量实现多态对齐策略，并在不影响用户配置的前提下完成底层对齐。注释清晰，设计决策值得参考。

功能与动机

O_DIRECT 要求内存缓冲区是页对齐的。原有的 CPU 卸载区域未做对齐，导致文件系统 tier 测试失败（见 PR body）。通过添加对齐机制解决，同时保持对用户 cpu_bytes_to_use 设定值的尊重。

实现拆解

1. 在 CPUOffloadingSpec 中添加对齐机制 - 新增类变量 BLOCK_SIZE_ALIGNMENT，默认值 1（不对齐）。 - 在 __init__ 中使用 round_up 将 kv_bytes_per_offloaded_block 对齐到 BLOCK_SIZE_ALIGNMENT 的倍数，并保存为 self.kv_bytes_per_offloaded_block。 - num_blocks 基于对齐后的块大小计算，确保不超过用户设置的 cpu_bytes_to_use。 - 涉及的调整还包括将 world_size 提前使用、在条件判断中加入 world_size > 0 避免除零。
2. 重构 SharedOffloadRegion 构造接口 - 移除 total_size_bytes 和 num_workers 参数，改为接收 kv_bytes_per_block（已对齐的每个块大小）。 - 内部推导 _row_stride = kv_bytes_per_block 和 total_size_bytes = num_blocks * _row_stride。 - 新增断言 kv_bytes_per_block % self.page_size == 0 确保对齐不变。 - 新增类变量 BLOCK_SIZE_ALIGNMENT: int = mmap.PAGESIZE，供上级 spec 覆写时引用。
3. TieringOffloadingSpec 覆写对齐常量 - 继承 CPUOffloadingSpec 并覆写 BLOCK_SIZE_ALIGNMENT = SharedOffloadRegion.BLOCK_SIZE_ALIGNMENT。 - 在 get_manager() 和 create_handlers() 中创建 SharedOffloadRegion 时，传入 kv_bytes_per_block=self.kv_bytes_per_offloaded_block（已对齐），符合新接口。 - 相应地移除了 world_size 和 num_workers 的重复计算。
4. 更新测试覆盖 - test_fs_tier.py：添加 _page_aligned_zero_tensor 和 _page_aligned_rand_tensor 辅助函数，保证 fixture 中的张量是页对齐的。移除了原来依赖 SharedOffloadRegion 的测试，保持测试独立性。 - test_shared_offload_region.py 和 test_gpu_worker.py：调整 SharedOffloadRegion 构造调用，使用新参数 kv_bytes_per_block。

关键文件：

- vllm/v1/kv_offload/cpu/spec.py（模块 CPU 卸载规格；类别 source；类型 core-logic；符号 BLOCK_SIZE_ALIGNMENT, aligned_kv_bytes_per_offloaded_block, kv_bytes_per_offloaded_block, cpu_page_size_per_worker）：核心变更文件：在 CPUOffloadingSpec 中添加对齐逻辑 (BLOCK_SIZE_ALIGNMENT, round_up)，调整 num_blocks 和 kv_bytes_per_offloaded_block 计算，确保对齐后不超出用户配置。
- tests/v1/kv_offload/tiering/test_fs_tier.py（模块 文件系统测试；类别 test；类型 test-coverage；符号 _page_aligned_zero_tensor, _page_aligned_rand_tensor, _BLOCK_ELEMENTS）：测试核心对齐：添加页对齐张量辅助函数，替换原有未对齐的 torch.zeros，确保文件系统测试在 O_DIRECT 下通过。
- vllm/v1/kv_offload/cpu/shared_offload_region.py（模块 共享卸载区域；类别 source；类型 core-logic；符号 BLOCK_SIZE_ALIGNMENT, kv_bytes_per_block, _row_stride）：SharedOffloadRegion 构造函数重构：移除 total_size_bytes/num_workers，改为接收对齐后的 kv_bytes_per_block，内部计算 row_stride 和 total_size_bytes，新增对齐断言。
- vllm/v1/kv_offload/tiering/spec.py（模块 分级卸载规格；类别 source；类型 dependency-wiring；符号 BLOCK_SIZE_ALIGNMENT）：TieringOffloadingSpec 覆写 BLOCK_SIZE_ALIGNMENT，并调整 SharedOffloadRegion 调用参数，适配新接口。
- tests/v1/kv_offload/cpu/test_shared_offload_region.py（模块 区域测试；类别 test；类型 test-coverage；符号 _make_region, _multi_region, _race_construct, _mp_race_construct_and_write）：适配 SharedOffloadRegion 新构造参数，移除 total_size_bytes/num_workers，使用 kv_bytes_per_block。
- tests/v1/kv_offload/cpu/test_gpu_worker.py（模块 GPU 工作测试；类别 test；类型 test-coverage；符号 cpu_page_size）：适配 SharedOffloadRegion 新构造参数，在 test_transfer 中使用 round_up 对齐 cpu_page_size 并传入 kv_bytes_per_block。

关键符号：CPUOffloadingSpec.init, SharedOffloadRegion.init, TieringOffloadingSpec.init, TieringOffloadingSpec.get_manager, TieringOffloadingSpec.create_handlers, _page_aligned_zero_tensor, _page_aligned_rand_tensor, _make_region, _multi_region, _race_construct, _mp_race_construct_and_write

关键源码片段

vllm/v1/kv_offload/cpu/spec.py

核心变更文件：在 CPUOffloadingSpec 中添加对齐逻辑 (BLOCK_SIZE_ALIGNMENT, round_up)，调整 num_blocks 和 kv_bytes_per_offloaded_block 计算，确保对齐后不超出用户配置。

```
# CPUOffloadingSpec 核心对齐逻辑（vllm/v1/kv_offload/cpu/spec.py）
class CPUOffloadingSpec(OffloadingSpec):
    BLOCK_SIZE_ALIGNMENT = 1 # 默认不对齐，子类可覆写为页大小

    def __init__(self, vllm_config: VllmConfig, kv_cache_config: KVCacheConfig):
        super().__init__(vllm_config, kv_cache_config)
```

```
        cpu_bytes_to_use = self.extra_config.get("cpu_bytes_to_use")
        if not cpu_bytes_to_use:
            raise Exception(
                "cpu_bytes_to_use must be specified in kv_connector_extra_config"
            )

        world_size = vllm_config.parallel_config.world_size
        self.num_blocks = 0
        self.kv_bytes_per_offloaded_block = 0
        self.cpu_page_size_per_worker = 0

        assert kv_cache_config is not None
        if kv_cache_config.num_blocks > 0 and world_size > 0:
            total_gpu_kv_bytes = sum(t.size for t in kv_cache_config.kv_cache_tensors)
            kv_bytes_per_block = (
                total_gpu_kv_bytes // kv_cache_config.num_blocks
            ) * world_size
            kv_bytes_per_offloaded_block = kv_bytes_per_block * self.block_size_factor

            # 每 worker 的页面大小（无对齐）
            self.cpu_page_size_per_worker = kv_bytes_per_offloaded_block // world_size

            # 将对齐后的块大小向上取整到 BLOCK_SIZE_ALIGNMENT 的倍数
            aligned_kv_bytes_per_offloaded_block = round_up(
                kv_bytes_per_offloaded_block, self.BLOCK_SIZE_ALIGNMENT
            )
            self.num_blocks = int(cpu_bytes_to_use) // aligned_kv_bytes_per_offloaded_block

            # 保存对齐后的值（可能包含 padding）
            self.kv_bytes_per_offloaded_block = aligned_kv_bytes_per_offloaded_block
        # 后续初始化 manager 等
```

tests/v1/kv_offload/tiering/test_fs_tier.py

测试核心对齐：添加页对齐张量辅助函数，替换原有未对齐的 torch.zeros，确保文件系统测试在 O_DIRECT 下通过。

```
# 测试辅助函数：创建页对齐的张量（tests/v1/kv_offload/tiering/test_fs_tier.py）
def _page_aligned_zero_tensor(
    num_blocks: int, block_elements: int, dtype: torch.dtype = _DTYPE
) -> torch.Tensor:
    page_size = mmap.PAGESIZE
    dtype_num_bytes = torch.tensor([], dtype=dtype).element_size()
    num_bytes = num_blocks * block_elements * dtype_num_bytes
    num_bytes_aligned = num_bytes + page_size # 多分配一页，便于后续偏移
    t = torch.zeros(num_bytes_aligned, dtype=torch.uint8)

    ptr = t.data_ptr()
    alignment_offset = ptr % page_size
    shift = page_size - alignment_offset # 偏移到下一个页边界
```

```
    t = t[shift : shift + num_bytes] # 取页对齐后的子区间
    return t.view(dtype).view(num_blocks, block_elements)


def _page_aligned_rand_tensor(
    num_blocks: int, block_elements: int, dtype: torch.dtype = _DTYPE
) -> torch.Tensor:
    rand_tensor = _page_aligned_zero_tensor(num_blocks, block_elements)
    rand_tensor[:] = torch.rand(num_blocks, block_elements, dtype=dtype)
    return rand_tensor
```

vllm/v1/kv_offload/cpu/shared_offload_region.py

SharedOffloadRegion 构造函数重构：移除 total_size_bytes/num_workers，改为接收对齐后的 kv_bytes_per_block，内部计算 row_stride 和 total_size_bytes，新增对齐断言。

```
# SharedOffloadRegion 构造函数（vllm/v1/kv_offload/cpu/shared_offload_region.py）
class SharedOffloadRegion:
    BLOCK_SIZE_ALIGNMENT: int = mmap.PAGESIZE # 页对齐常量，供上级 spec 引用

    def __init__(
        self,
        instance_id: str,
        num_blocks: int,
        rank: int | None,
        kv_bytes_per_block: int, # 已对齐到 page_size 的每个块大小
        cpu_page_size: int,
    ) -> None:
        self.page_size = mmap.PAGESIZE
        assert kv_bytes_per_block % self.page_size == 0 # 确保对齐

        self.num_blocks = num_blocks
        self._row_stride = kv_bytes_per_block # 每行一个块（所有 worker 的数据连续存放）
        self.total_size_bytes = self.num_blocks * self._row_stride
        # 后续创建 mmap 文件与映射
```

评论区精华

1. 对齐位置选择：orozery 指出直接在 SharedOffloadRegion 中对齐可能违反用户 cpu_bytes_to_use，建议将对齐逻辑上移到 CPUOffloadingSpec，通过类变量控制。最终采用方案。
2. 用户配置尊重：varun 担心对齐后块变大导致总块数减少，orozery 认为合理，用 round_up 后整除 cpu_bytes_to_use 来限制。
3. 测试独立性：orozery 希望 test_fs_tier.py 不依赖 SharedOffloadRegion，使用简单的页对齐张量即可。作者实现 _page_aligned_zero_tensor 满足要求。

- 对齐位置选择：在 CPUOffloadingSpec 还是 SharedOffloadRegion 中对齐？ (design): 最终采用类变量方案：CPUOffloadingSpec 定义 BLOCK_SIZE_ALIGNMENT=1，TieringOffloadingSpec 覆写为 mmap.PAGESIZE；对齐在 spec 层完成，

SharedOffloadRegion 接收已对齐的值。

- 如何确保不超出用户设置的 cpu_bytes_to_use？ (correctness): 使用 round_up 后赋值给 aligned_kv_bytes_per_offloaded_block，再用 int(cpu_bytes_to_use) // aligned_kv_bytes_per_offloaded_block 计算块数，确保总和不超过用户设定。
- 文件系统测试是否应保持独立？ (testing): 移除 test_fs_tier.py 中对 SharedOffloadRegion 的使用，添加 _page_aligned_zero_tensor 和 _page_aligned_rand_tensor 辅助函数。

风险与影响

- 风险：
  1. 内存浪费：对齐可能导致每个块末尾少量 padding，但相对于总内存可忽略。
  2. 接口破坏：SharedOffloadRegion 构造函数参数已改变，但该模块仅 v1 内部使用，不影响外部用户。
  3. 新增断言：kv_bytes_per_block % page_size == 0 可能在新场景中触发，但有助于及时暴露对齐问题。
  4. 测试覆盖：对齐逻辑在 CPUOffloadingSpec 和 SharedOffloadRegion 中均有测试，但 CPUOffloadingSpec 的对齐行为依赖 BLOCK_SIZE_ALIGNMENT 默认值，子类覆写时需要额外测试。 - 影响：影响范围限定在 vllm v1 的 KV offload 模块，具体为文件系统 tier 和共享卸载区域。用户无感知，但运行时可靠性提升。开发团队需注意后续创建 SharedOffloadRegion 时使用新接口。 - 风险标记：对齐 padding 可能轻微浪费内存，SharedOffloadRegion 构造接口破坏性变更，新断言可能触发边界情况

关联脉络

- PR #44287 [KV Offloading] Enable HMA models for Tiering Offloading: 都修改了 vllm/v1/kv_offload/tiering/spec.py，属于同一 KV offload 功能线的连续演进。
- PR #44293 Nit Changes in Tiered KV Offload: 修改了 tests/v1/kv_offload/tiering/test_fs_tier.py 等文件，与当前 PR 的测试文件重叠，体现同一模块的增量改进。