PR #33666 完整报告

sgl-project/sglang

fix(PP): size the mamba pool per pipeline stage, not per whole model

合并时间：2026-08-07 04:10

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33666

执行摘要

- 一句话：修复 PP 下 mamba 池按整模型计费的容量 bug
- 推荐动作：值得精读。核心价值不在代码量，而在 PR body 的设计论证：如何在不引入集合通信的前提下让所有 PP rank 对共享容量参数达成一致（取最大阶段份额），以及用 145 预算扫描定量证明方案完备性。对于从事容量规划、调度器或多卡一致性工作的工程师，这是很好的案例。建议重点阅读 _handle_max_mamba_cache 的 auto-fit 与显式容量两条分支如何统一改口径，以及 TestPPMambaPoolSizing 的两个断言如何精准卡住回归。

功能与动机

PR body 指出：KVCacheConfigurator._handle_max_mamba_cache 依据 config.mamba2_cache_params.mamba_cache_per_req 计费，而该值来自整模型 mamba 层列表；但 “Under pipeline parallelism each rank only allocates state for the layers in its own [start_layer, end_layer) slice”，因此每个阶段被按约 pp_size 倍的成本高估。具体到 Kimi-K3（93 层、69 个线性注意力层）pp_size=8：池只有 26 槽，resolve_max_num_reqs 随之把 max_running_requests 钳到 6，并进一步传导到 pp_max_micro_batch_size。显式 --max-mamba-cache-size 场景同样受害：求解器按整模型每请求成本记账，328 GiB 预算下 KV 预算为负、启动失败。修复目标是让容量计算与每阶段真实分配一致，同时保证各 rank 推导结果统一。

实现拆解

变更入口：KVCacheConfigurator._handle_max_mamba_cache（python/sglang/srt/mem_cache/kv_cache_configurator.py），该函数负责根据剩余显存预算求解 mamba 状态池容量 max_mamba_cache_size。

1. 按 PP 阶段缩放每请求成本：新增 get_pp_indices 导入；在函数开头读取全部 mamba 层列表 all_mamba_layers，当 pp_size > 1 时遍历所有 rank 的 [start, end) 分片，统计每个阶段实际持有的 mamba 层数并取最大值，得到 pp_layer_scale = max_stage_mamba_layers / len(all_mamba_layers)，由此计算 stage_per_req = int(mamba_cache_per_req * pp_layer_scale)。取最大阶段份额而非本 rank 份额，是为了让每个 rank 本地推导出相同数值、避免引入集合通信，同时避免 69 层在 8 阶段不均匀分割时 max_running_requests 与 pp_max_micro_batch_size 跨阶段分歧。
2. 同步修改所有计费点：后续 3 处原引用 mamba_cache_per_req 的位置全部改为 stage_per_req——spec-dec 中间态 intermediate_size（显式容量分支与 capped 分支）

与 auto-fit 分支的 per_req；同时 replayssm_ring_per_req 也乘以 pp_layer_scale，保证 ReplaySSM ring 与池容量等比。pp_size=1 时 pp_layer_scale 恒为 1，行为完全不变。

3. 配套测试：test_mamba_donated_alloc_ratio.py 新增 TestPPMambaPoolSizing，用 SimpleNamespace 伪造 KVCacheConfigurator 依赖（Kimi-K3 形状、8 GiB 预算），通过 runtime_context.override_server_args 禁掉显式容量参数后调用 _handle_max_mamba_cache 并读取结果。两个用例分别验证"阶段不再按整模型计费"（staged > solo * 5）与"8 个 PP rank 池大小全部一致"。测试纯 CPU、无权重依赖，未打补丁时第一个用例失败（1 failed, 7 passed → 8 passed）。
4. 端到端验证（非代码改动）：Kimi-K3 2 节点 pp_size=8 实测池 26→210 槽、max_running_requests 6→52，GSM8K 200 题精度 0.985→0.990；1P1D 98,827 请求 trace 下并发 24 时达 78,437 tok/s（每 GPU 4,902 tok/s）。

关键文件：

- python/sglang/srt/mem_cache/kv_cache_configurator.py（模块 缓存配置；类别 source；类型 core-logic；符号 _handle_max_mamba_cache）：核心修复文件：_handle_max_mamba_cache 中把每请求 mamba 成本从整模型口径改为按 PP 最大阶段份额缩放，并同步修正 ReplaySSM ring 与 spec-dec 中间态计费。
- test/registered/unit/mem_cache/test_mamba_donated_alloc_ratio.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestPPMambaPoolSizing, _pool_size, test_stage_is_not_charged_for_the_whole_model, test_every_stage_agrees_on_the_pool_size）：新增 TestPPMambaPoolSizing 回归测试，用 CPU-only fake 对象验证 PP 阶段池容量放大与跨 rank 一致性，未打补丁时第一个用例失败。

关键符号：_handle_max_mamba_cache, TestPPMambaPoolSizing._pool_size, test_stage_is_not_charged_for_the_whole_model, test_every_stage_agrees_on_the_pool_size

关键源码片段

python/sglang/srt/mem_cache/kv_cache_configurator.py

核心修复文件：_handle_max_mamba_cache 中把每请求 mamba 成本从整模型口径改为按 PP 最大阶段份额缩放，并同步修正 ReplaySSM ring 与 spec-dec 中间态计费。

```
# python/sglang/srt/mem_cache/kv_cache_configurator.py
# KVCacheConfigurator._handle_max_mamba_cache 中与 PP 缩放相关的改动点
def _handle_max_mamba_cache(self, total_rest_memory):
    config = self.mambaish_config
    server_args = self.server_args
    assert config is not None

    # 原逻辑用整模型 mamba 层列表计算每请求成本；PP 下每个 rank 只分配
    # 自己 [start_layer, end_layer) 分片内的层状态，因此按整模型计费会把
    # 池容量压到实际可容纳量的大约 1/pp_size。
    all_mamba_layers = config.mamba2_cache_params.layers
    if self.ps.pp_size > 1 and all_mamba_layers:
```

```
        # 取所有阶段中持有 mamba 层数最多的份额，而不是本 rank 自己的份额：
        # 前者让每个 rank 本地推导出相同的池容量（无需集合通信），且系统容量
        # 本来就由最重的阶段决定。
        max_stage_mamba_layers = max(
            sum(1 for i in all_mamba_layers if start <= i < end)
            for start, end in (
                get_pp_indices(
                    self.model_config.num_hidden_layers, rank, self.ps.pp_size
                )
                for rank in range(self.ps.pp_size)
            )
        )
    else:
        # pp_size=1 时本 rank 持有全部层，scale 恒为 1，行为与之前完全一致。
        max_stage_mamba_layers = len(all_mamba_layers)

    pp_layer_scale = max_stage_mamba_layers / max(len(all_mamba_layers), 1)
    # 每请求状态成本改为阶段口径，后续所有预算计费点都使用 stage_per_req。
    stage_per_req = int(
        config.mamba2_cache_params.mamba_cache_per_req * pp_layer_scale
    )

    # 中间 has_spec_dec、ReplaySSM 分支等未改动逻辑省略，以下展示三个使用点。

    # ReplaySSM ring 与池等比例缩放，避免环形缓冲相对池容量错配
    replayssm_ring_per_req = int(replayssm_ring_per_req * pp_layer_scale)

    # 显式 --max-mamba-cache-size 分支：spec-dec 中间态同样改用阶段成本
    intermediate_size = (
        stage_per_req
        * (get_schedule().max_mamba_cache_size + 1)
        * get_spec().speculative_num_draft_tokens
    )

    # auto-fit 分支：用阶段每请求成本解出池大小，替代原来的整模型成本
    assert stage_per_req > 0
    per_req = stage_per_req
```

test/registered/unit/mem_cache/test_mamba_donated_alloc_ratio.py

新增 TestPPMambaPoolSizing 回归测试，用 CPU-only fake 对象验证 PP 阶段池容量放大与跨 rank 一致性，未打补丁时第一个用例失败。

```
# test/registered/unit/mem_cache/test_mamba_donated_alloc_ratio.py
class TestPPMambaPoolSizing(unittest.TestCase):
    """PP 下每个 rank 只分配自己的 [start_layer, end_layer) 分片 mamba 状态，
    因此按整模型层数计费会饿死池子。正确做法是取各阶段中最大的层份额，
    这样所有 rank 的池大小（进而 max_running_requests 与
    pp_max_micro_batch_size）一致，且无需集合通信。"""
```

```
# Kimi-K3 形状：93 层，除每第 4 层和最后一层外都是线性注意力，
# 69 个 mamba 层在 8 个阶段上不均匀分割（每阶段 9 或 8 个）。
TOTAL_LAYERS = 93
MAMBA_LAYERS = [i for i in range(93) if (i + 1) % 4 != 0 and i <= 90]
BUDGET_GB = 8.0

@classmethod
def _pool_size(cls, pp_rank, pp_size):
    # 用 SimpleNamespace 伪造 configurator 依赖，纯 CPU 跑容量求解，
    # 并通过 runtime_context 覆盖 server args，最后读取 schedule 上的池大小。
    from sglang.srt import runtime_context as rc
    from sglang.srt.configs.mamba_utils import (
        Mamba2CacheParams,
        Mamba2StateDType,
        Mamba2StateShape,
    )
    from sglang.srt.distributed.utils import get_pp_indices
    from sglang.srt.mem_cache.kv_cache_configurator import KVCacheConfigurator
    from sglang.srt.runtime_context import get_schedule

    shape = Mamba2StateShape(
        conv=[(4096, 3)],
        temporal=(64, 128, 128),
        intermediate_size=0,
        conv_dim=0,
        ssm_state_size=0,
        num_heads=0,
        head_dim=0,
        state_size=0,
        conv_kernel=0,
        num_k_heads_per_tp=8,
    )
    params = Mamba2CacheParams(
        shape=shape,
        dtype=Mamba2StateDType(conv=torch.bfloat16, temporal=torch.float32),
        layers=list(cls.MAMBA_LAYERS),
    )
    start, end = get_pp_indices(cls.TOTAL_LAYERS, pp_rank, pp_size)
    fake = SimpleNamespace(
        mambaish_config=SimpleNamespace(mamba2_cache_params=params),
        server_args=SimpleNamespace(),
        spec_algorithm=SimpleNamespace(is_none=lambda: True),
        layer_info=SimpleNamespace(start_layer=start, end_layer=end),
        ps=SimpleNamespace(attn_dp_size=1, pp_size=pp_size),
        hybrid_gdn_config=None,
        model_config=SimpleNamespace(
            hf_config=SimpleNamespace(), num_hidden_layers=cls.TOTAL_LAYERS
        ),
    )
```

```
        with rc.get_context().override_server_args(
            disable_radix_cache=False,
            max_mamba_cache_size=None,
            max_running_requests=None,
            mamba_full_memory_ratio=0.5,
            enable_linear_replayssm_spec=False,
        ):
            KVCacheConfigurator._handle_max_mamba_cache(fake, cls.BUDGET_GB)
            return get_schedule().max_mamba_cache_size

    def test_stage_is_not_charged_for_the_whole_model(self):
        solo = self._pool_size(0, 1)
        staged = self._pool_size(0, 8)
        # 最忙的阶段持有 69 层中的 9 层，池容量应比整模型计费时大 5 倍以上。
        self.assertGreater(staged, solo * 5)

    def test_every_stage_agrees_on_the_pool_size(self):
        sizes = {self._pool_size(r, 8) for r in range(8)}
        self.assertEqual(
            len(sizes), 1, f"per-rank pool sizes diverged: {sorted(sizes)}"
        )
```

评论区精华

该 PR 无 review 评论，ispobock 直接 APPROVE；核心设计论证全部写在 PR body 的 Modifications 一节。最值得注意的权衡是“按最大阶段份额缩放”而非“按本 rank 自己的份额”：作者对 145 种 mamba 预算（4–40 GiB）做扫描，发现按本 rank 份额计算时 95/145（65.5%）预算会让 pp_max_micro_batch_size 跨阶段不一致，而最大份额方案在 145 种预算下全部一致（0/145）。PR body 强调：pp_max_micro_batch_size 决定批次如何切成 micro-batch，阶段之间一旦不一致，邻居阶段会对“哪些序列在飞”产生分歧，因此该值必须在所有阶段一致。选择最大阶段份额的本质是：系统容量总由最重的阶段决定，而取最大值恰好让每个 rank 从 get_pp_indices 本地算出同一结果。

- 为什么用最大阶段份额而不是本 rank 自己的份额 (design): 采用持有最多 mamba 层的阶段的份额作为缩放基准；每个 rank 通过 get_pp_indices 本地推导，无需集合通信。

风险与影响

- 风险：
  - 显存占用显著上升：示例中池从 26 槽升至 210 槽，ssm_state 实际占用随之增长；此前依赖旧隐性限制规避显存压力的部署需要重新核对显存预算。显式 --max-mamba-cache-size 场景下 KV 池预算分配变化较大（328 GiB 时从 -19.20 GiB 变为 +58.37 GiB），属于修复方向，但运维预期需要更新。
  - 并发上限提升 8.7 倍会放大调度与网络压力：max_running_requests 从 6 到 52，prefill 队列、KV 池与负载均衡都按新容量工作，需确认下游容量配套。
  - 均匀切分假设：代码假定层按 get_pp_indices 均匀分配；若未来引入非均匀或跳过层分配，需要同步此计算口径。

- 测试覆盖局限：新增测试用伪造对象驱动 _handle_max_mamba_cache，未覆盖真实多进程握手路径，也未覆盖 spec-dec 与 ReplaySSM 组合分支。
- 范围控制：pp_size=1、非 hybrid 模型完全不受影响，回归面有限。

- 影响：
  - 用户：Kimi-K3 等混合线性注意力模型在 PP 部署下的并发与吞吐显著提升（实测并发上限 8.7 倍、trace 吞吐最高 78,437 tok/s），并消除“调大 mamba 缓存反而启动失败”的反直觉行为。
  - 系统：max_mamba_cache_size、max_running_requests、pp_max_micro_batch_size 在各 PP 阶段间保持一致，消除了容量配置层面的隐性分歧源。
  - 团队：修复容量规划口径错误，为后续线性注意力模型（Kimi、MiniMax-H3、Grok 等）的 PP 部署提供正确基准；新增测试锁定该回归，后续改动 kv_cache_configurator.py 时会被 CI 覆盖。
  - 影响程度：中高，涉及核心调度容量路径，但逻辑改动小、pp_size=1 分支不变，风险可控。
  - 风险标记：核心路径变更，显存占用增加，测试未覆盖真实启动路径，并发上限提升可能放大下游负载

关联脉络

- PR #27010 [HiCache] Fix PP inconsistency with HiCache L3 (#22607): 同为 PP 场景下跨阶段一致性修复，涉及缓存与调度容量路径，可对照理解 PP 下容量与一致性问题的处理模式。
- PR #36219 [Performance] Tune FlashInfer EXTEND for DP prefill: 同为容量与调度路径的性能调优（DP prefill 缓冲预热），与本 PR 释放的池容量共同作用于 max_running_requests 与吞吐上限。