PR #24575 完整报告

sgl-project/sglang

[NPU] add zbal support for npu

合并时间：2026-05-13 20:52

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24575

执行摘要

- 一句话：NPU 后端集成 zbal 零缓冲区加速库
- 推荐动作：该 PR 值得 NPU 用户和关心内存优化的工程师精读，尤其是 lazy_init_zbal_gva_mem 的混合分配策略。建议在合并后补充 NPU 环境下的性能基准测试文档。

功能与动机

zbal 是基于 memfabric 的零缓冲区加速库，已在 Ascend NPU 上开源。PR body 中称其在 w8a8 & 4k per device in DSV3 like moe process 上实现了 30%+ 的加速和 1.2G+ 的内存效率提升。通过集成 zbal，NPU 用户可以显著提升模型推理性能并优化内存利用率。

实现拆解

1. 环境变量注册：在 python/sglang/srt/environ.py 中添加 SGLANG_ZBAL_LOCAL_MEM_SIZE 和 SGLANG_ZBAL_BOOTSTRAP_URL 两个环境变量，分别控制 zbal 分配的内存大小（MB）和分布式引导 URL。
2. 初始化入口：在 python/sglang/srt/managers/scheduler.py 的 __init__ 中新增 init_zbal_on_npu() 调用，放在 IPC 通道初始化之后、任何 PyTorch 分配之前。新建方法 init_zbal_on_npu 在 NPU 设备上调用 sglang.srt.hardware_backend.npu.utils.init_zbal。
3. 核心 init / lazy_init：在 python/sglang/srt/hardware_backend/npu/utils.py 中新增 init_zbal 和 lazy_init_zbal_gva_mem 两个函数。init_zbal 若为 mix alloc 模式则仅调用 switch_to_allocator() 并返回；否则直接通过 zbal_init 初始化全部 GVA 空间。lazy_init_zbal_gva_mem 在权重和 KV cache 加载完毕（即 model_runner 的 initialize 中）调用，计算剩余可用内存并动态分配 GVA 空间，实现混合分配。
4. 模型执行器适配：在 python/sglang/srt/model_executor/model_runner.py 的 initialize 中将 NPU 分支从 elif self.device in ["npu", "cpu"] 拆出单独处理，在 attention 后端初始化之后、设备图捕获之前插入 lazy_init_zbal_gva_mem 调用。
5. 内存查询改写：在 python/sglang/srt/utils/common.py 中修改 get_available_gpu_memory 和 get_npu_memory_capacity，当启用 zbal 时调用 zbal 自己的内存查询接口（zbal.zbal_module.mem_get_info）或直接返回预设值。
6. DeepEP 适配：在 python/sglang/srt/layers/moe/token_dispatcher/deepep.py 中根据是否启用 zbal 条件导入 zbal.zbal.deepep_adaptor 和 zbal.zbal_buffer 替代原 deep_ep。

关键文件：

- python/sglang/srt/hardware_backend/npu/utils.py（模块 NPU 工具；类别 source；类型 core-logic；符号 init_zbal, lazy_init_zbal_gva_mem）：核心修改文件，新增 init_zbal 和 lazy_init_zbal_gva_mem 函数，实现 zbal 的初始化和延迟分配逻辑。
- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 init_zbal_on_npu）：调度器初始化入口，新增 init_zbal_on_npu 方法，确保在 torch 分配前切换分配器。
- python/sglang/srt/model_executor/model_runner.py（模块 模型执行器；类别 source；类型 data-contract）：模型执行器初始化，将 NPU 分支拆出并在 attention 后端初始化后、CUDA graph 捕获前调用 lazy_init_zbal_gva_mem。

关键符号：init_zbal, lazy_init_zbal_gva_mem, init_zbal_on_npu, get_available_gpu_memory, get_npu_memory_capacity

关键源码片段

python/sglang/srt/managers/scheduler.py

调度器初始化入口，新增 init_zbal_on_npu 方法，确保在 torch 分配前切换分配器。

```
# python/sglang/srt/managers/scheduler.py

class Scheduler:
    def __init__(self, ...):
        # ... 其他初始化
        self.init_ipc_channels(port_args)

        # Init ZBAL，切换分配器必须发生在任何 torch 分配之前
        self.init_zbal_on_npu()

        # Init PD-multiplexing context
        if self.enable_pdmux:
            self.init_pdmux()
        # ...

    def init_zbal_on_npu(self):
        if _is_npu:
            from sglang.srt.hardware_backend.npu.utils import init_zbal

            if self.pp_size > 1:
                logger.error(f"only zbal mix mode support pp_size > 1!")
            init_zbal(
                self.tp_size, self.gpu_id, self.tp_rank
            ) # 如果是混合模式，仅切换分配器
```

python/sglang/srt/model_executor/model_runner.py

模型执行器初始化，将 NPU 分支拆出并在 attention 后端初始化后、CUDA graph 捕获前调用 lazy_init_zbal_gva_mem。

```
# python/sglang/srt/model_executor/model_runner.py

class ModelRunner:
    def initialize(self, pre_model_load_memory: float):
        # ... 前面初始化
        elif self.device == "npu":
            self.init_attention_backend()
            # lazy init for zbal with mix mode ( 必须在 CUDA graph 捕获之前 )
            if envs.SGLANG_ZBAL_LOCAL_MEM_SIZE.get() > 0 and not self.is_draft_worker:
                from sglang.srt.hardware_backend.npu.utils import lazy_init_zbal_gva_mem

                lazy_init_zbal_gva_mem(
                    self.device,
                    self.gpu_id,
                    get_world_group().rank_in_group,
                    get_world_group().world_size,
                    get_world_group().cpu_group,
                )
            self.init_device_graphs()
```

评论区精华

PR 的 review 评论较少，主要由 sglang-npu-bot 触发 CI 测试并最终批准。提交历史显示作者在多次合并 main 分支后进行了代码清理和 lint 应用，整体改动较为收敛，未出现重大设计争议。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 依赖风险：引入外部库 zbal，该库只在 Ascend NPU 环境下可用，若缺失则导入失败，但代码通过条件导入（仅在 NPU 且环境变量开启时尝试）隔离，不会阻塞其他硬件平台。
  2. 兼容性：修改了 get_available_gpu_memory 和 get_npu_memory_capacity 的行为，当启用 zbal 时返回 zbal 管理的显存量而非实际物理显存，可能影响依赖这些函数进行内存预算的其他模块（如 cache 分配）。但该修改受环境变量开关保护，默认关闭。
  3. 初始化顺序敏感：init_zbal_on_npu 必须在任何 torch 分配之前调用，在 __init__ 中已安排在 IPC 初始化之后、其他 init 之前，时序正确。lazy_init_zbal_gva_mem 在权重和 KV 加载后调用，需确保此时显存已分配完毕。
  4. 非 NPU 平台：所有改动均包含 _is_npu 或环境变量守卫，不影响其他后端。
  5. 测试覆盖不足：源码改动未包含对应测试文件，仅通过 CI 进行集成验证，缺少单元测试。
     - 影响：影响范围限于 NPU 后端用户，通过环境变量自愿开启，默认关闭。对于不设置 SGLANG_ZBAL_LOCAL_MEM_SIZE 的用户，行为完全不变。对于开启的用户，zbal 将接管显存分配，可能提升 MoE 类模型的吞吐和显存效率，但需要用户正确配置内存大小并理解 zbal 的工作模式（mix alloc 或全 GVA）。对团队而言，该 PR 是 NPU 生态建设的重要一步，为后续更深入的硬件加速铺平道路。 - 风险标记：依赖外部库 zbal，初始化

顺序敏感，内存查询行为变更，缺少单元测试覆盖

关联脉络

- PR #25114 [NPU] [DOC] add performance testing and optimization docs for npu: 同属于 NPU 生态建设系列，该文档可为 zbal 的使用提供性能测试参考。