PR #20816 完整报告

sgl-project/sglang

[Diffusion][CPU] Init CPU platform support for SGLang Diffusion

合并时间：2026-04-21 14:25

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/20816

执行摘要

- 一句话：为 SGLang Diffusion 添加原生 CPU 平台支持，实现纯 CPU 推理和优化绑定。
- 推荐动作：该 PR 值得精读，特别是关注 CPUWorker 继承设计和共享内存通信优化，这些设计决策展示了如何扩展平台支持并保持代码一致性。

功能与动机

PR body 明确指出目的是扩展 SGLang Diffusion 到仅 CPU 平台（例如 Intel Xeon），以提供无 GPU 依赖的部署选项。动机源于支持更多硬件环境和降低部署门槛。

实现拆解

1. 创建 PyTorch 原生回退函数：新增文件 python/sglang/jit_kernel/diffusion/triton/torch_fallback.py，定义关键操作如 fuse_scale_shift_kernel_native、apply_rotary_embedding_native，作为 CPU、MPS 和 NPU 平台的共享纯 PyTorch 实现，替代 Triton 内核。
2. 实现 CPU 平台逻辑：修改 python/sglang/multimodal_gen/runtime/platforms/cpu.py，添加方法如 get_local_torch_device、get_attn_backend_cls_str（默认使用 Torch SDPA 后端）和 enable_dit_layerwise_offload_for_wan_by_default（禁用分层卸载）。
3. 引入 CPUWorker 类：新增文件 python/sglang/multimodal_gen/runtime/managers/cpu_worker.py，定义 CPUWorker 继承自 GPUWorker，覆写 __init__ 方法添加 init_cpu_threads_binding，处理 OMP 线程绑定和 NUMA 节点分配。
4. 集成到调度系统：修改 python/sglang/multimodal_gen/runtime/managers/scheduler.py，基于平台检测动态选择 CPUWorker 或 GPUWorker，确保 CPU 路径入口。
5. 优化通信和配置：修改 python/sglang/multimodal_gen/runtime/distributed/group_coordinator.py，为 CPU 平台添加共享内存优化的 all_reduce 和 all_gather 路径；调整多个文件如 server_args.py 和 text_encoder_loader.py 以适配 CPU 逻辑。

关键文件：

- python/sglang/jit_kernel/diffusion/triton/torch_fallback.py（模块 内核回退；类别 source；类型 core-logic；符号 fuse_scale_shift_kernel_native, _expand, apply_rotary_embedding_native, norm_infer_native）：新增核心回退函数，为 CPU、MPS 和 NPU 平台提供纯 PyTorch 实现，是 CPU 支持的基础逻辑。

- python/sglang/multimodal_gen/runtime/managers/cpu_worker.py（模块 工作器管理；类别 source；类型 core-logic；符号 CPUWorker, init, init_cpu_threads_binding, _）：新增 CPUWorker 类，作为 CPU 平台的工作器实现，处理线程绑定和初始化逻辑。
- python/sglang/multimodal_gen/runtime/platforms/cpu.py（模块 平台抽象；类别 source；类型 core-logic；符号 get_local_torch_device, get_attn_backend_cls_str, enable_dit_layerwise_offload_for_wan_by_default）：修改 CPU 平台类，添加平台特定方法如设备获取和注意力后端配置。
- python/sglang/multimodal_gen/runtime/distributed/group_coordinator.py（模块 分布式通信；类别 source；类型 dependency-wiring）：修改通信逻辑，为 CPU 平台添加共享内存优化的 all_reduce 和 all_gather 路径。

关键符号：fuse_scale_shift_kernel_native, apply_rotary_embedding_native, norm_infer_native, init_cpu_threads_binding, get_attn_backend_cls_str

关键源码片段

python/sglang/jit_kernel/diffusion/triton/torch_fallback.py

新增核心回退函数，为 CPU、MPS 和 NPU 平台提供纯 PyTorch 实现，是 CPU 支持的基础逻辑。

```
def fuse_scale_shift_kernel_native(
    x: torch.Tensor,
    scale: torch.Tensor,
    shift: torch.Tensor,
    scale_constant: float = 1.0,
    block_l: int = 128,
    block_c: int = 128,
):
    # 原生回退函数，用于融合缩放和移位操作，支持 scale_constant 参数
    B, L, C = x.shape

    def _expand(t: torch.Tensor) -> torch.Tensor:
        # 辅助函数：根据输入张量维度扩展形状以匹配目标
        if t.dim() == 4:
            # 从 [B, F, 1, C] 扩展到 [B, L, C]
            num_frames = t.shape[1]
            frame_seqlen = L // num_frames
            return (
                t.squeeze(2)
                .unsqueeze(2)
                .expand(-1, -1, frame_seqlen, -1)
                .reshape(B, L, C)
            )
        elif t.dim() == 2:
            # 从 [B, C] 扩展到 [B, 1, C]
            return t.unsqueeze(1)
        return t
```

```
    scale = _expand(scale) # 统一扩展缩放张量
    shift = _expand(shift) # 统一扩展移位张量
    return x * (scale_constant + scale) + shift # 计算最终输出
```

python/sglang/multimodal_gen/runtime/managers/cpu_worker.py

新增 CPUWorker 类，作为 CPU 平台的工作器实现，处理线程绑定和初始化逻辑。

```
class CPUWorker(GPUWorker):
    # CPU 平台工作器，继承自 GPUWorker 以重用基础逻辑
    def __init__(
        self,
        local_rank: int,
        rank: int,
        master_port: int,
        server_args: ServerArgs,
    ):
        super().__init__(local_rank, rank, master_port, server_args) # 调用父类初始化
        if _is_cpu_amx_available:
            self.init_cpu_threads_binding() # 若支持 AMX，则初始化 CPU 线程绑定

    def init_cpu_threads_binding(self):
        # 初始化 CPU 线程绑定，基于环境变量和 NUMA 节点分配核心
        omp_cpuids = os.environ.get("SGLANG_CPU_OMP_THREADS_BIND", "all")
        cpu_ids_by_node = get_cpu_ids_by_node()
        n_numa_node = len(cpu_ids_by_node)
        if omp_cpuids == "all":
            # 默认绑定逻辑：每个 TP rank 使用一个 NUMA 节点的所有核心
            assert self.server_args.tp_size <= n_numa_node, (
                f"SGLANG_CPU_OMP_THREADS_BIND未设置时，tp_size必须小于等于NUMA节点数"
            )
            if self.server_args.tp_size < n_numa_node:
                logger.warning(f"仅使用部分NUMA节点")
            self.local_omp_cpuid = cpu_ids_by_node[self.rank]
        else:
            # 显式绑定逻辑：用户通过环境变量指定核心列表
            threads_bind_list = omp_cpuids.split("|")
            assert self.server_args.tp_size == len(threads_bind_list), (
                f"环境变量设置必须与TP大小对齐"
            )
            self.local_omp_cpuid = threads_bind_list[self.rank]
        torch.ops.sgl_kernel.init_cpu_threads_env(self.local_omp_cpuid) # 应用绑定
        os.environ["LOCAL_SIZE"] = str(self.server_args.tp_size) # 设置共享内存提示
        torch.ops.sgl_kernel.initialize(self.server_args.tp_size, self.rank) # 初始化内核
```

评论区精华

核心讨论围绕代码重构和设计权衡展开。

- 回退函数统一：mingfeima 建议将 CPU、MPS 和 NPU 的回退函数合并到 torch_fallback.py，避免重复代码，结论是采纳此建议以实现简化。
- CPUWorker 继承结构：mickqian 指出初始版本代码重复，建议继承 GPUWorker，最终实现中 CPUWorker 继承 GPUWorker 并仅覆写必要方法如 init_cpu_threads_binding。
- 注意力后端选择：mingfeima 询问 CPU 使用的注意力后端，在代码中明确为 Torch SDPA 后端。
- 共享内存通信：讨论中确认 CPU 平台优先使用共享内存优化的 shm_allreduce 和 shm_allgather，并在不可用时回退到标准 PyTorch 分布式操作。
- 回退函数统一设计 (design)：采纳建议，创建统一 torch_fallback.py 文件，简化了维护和导入逻辑。
- CPUWorker 继承结构 (design)：实现中 CPUWorker 继承 GPUWorker，仅覆写必要方法如 init_cpu_threads_binding，优化了代码结构。
- CPU 注意力后端选择 (correctness)：在 cpu.py 中明确设置 get_attn_backend_cls_str 返回 SDPA 后端，确保功能正确性。

风险与影响

- 风险：技术风险包括：
  1. 性能风险：PyTorch 原生回退函数（如 norm_infer_native）可能比 Triton 内核慢，影响 CPU 推理效率，需后续优化（如计划中的 C++ 内核）。
  2. 兼容性风险：CPU 架构差异（如 x86 vs ARM）可能导致绑定或内存计算错误，特别是在 get_cpu_ids_by_node 函数中。
  3. 测试覆盖不足：PR 未包含新的测试文件，可能遗漏 CPU 特定场景的回归测试。
  4. 共享内存依赖：group_coordinator.py 中的共享内存优化假设 intra-node 环境，若环境不满足可能触发错误回退。
- 影响：对用户、系统和团队的影响：
  - 用户影响：扩展 SGLang Diffusion 到无 GPU 环境，支持更广泛的部署场景（如边缘设备或低成本服务器）。
  - 系统影响：引入新的平台路径，增加代码复杂度，但通过继承和共享回退函数最小化维护开销。
  - 团队影响：为后续 CPU 优化（如 AMX 注意力后端）奠定基础，促进多平台开发流程。
  - 风险标记：性能回退风险，测试覆盖不足，平台兼容性

关联脉络

- 暂无明显关联 PR