PR #22842 完整报告

sgl-project/sglang

[CPU] Add gemma4_rmsnorm_cpu kernel

合并时间：2026-04-17 13:03

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/22842

执行摘要

- 一句话：新增 Gemma 4 RMSNorm CPU 内核，并扩展其他归一化内核支持 3D 输入，修复 Xeon CI 失败。
- 推荐动作：建议负责 CPU 内核开发或模型推理优化的工程师精读此 PR，重点关注 sgl-kernel/csrc/cpu/norm.cpp 中的 3D 输入支持设计和 python/sglang/srt/layers/layernorm.py 中的条件调用决策，这些设计模式在优化 CPU 计算和平衡性能与健壮性时值得借鉴。

功能与动机

根据 PR body，动机是修复 Xeon CI 失败（链接：https://github.com/sgl-project/sglang/actions/runs/24432405372/job/71379482058），该失败发生在 PR #21734 之后。通过添加新内核并扩展 3D 输入支持来解决此问题。

实现拆解

1. 添加 gemma4_rmsnorm_cpu 内核：在 sgl-kernel/csrc/cpu/norm.cpp 中实现新内核，支持 Gemma 4 模型的 RMSNorm 计算，参数包括 scale_shift 和 with_scale。
2. 扩展现有内核支持 3D 输入：修改 norm.cpp 中的 l2norm_kernel_impl 和 rmsnorm_kernel_impl，从仅处理 2D 输入（batch_size, hidden_size）扩展为处理 3D 输入（batch_size, seq_len, hidden_size），添加 stride 参数如 input_strideB 和 input_strideS 以支持任意形状张量。
3. Python 层集成：在 python/sglang/srt/layers/layernorm.py 中添加 forward_cpu 方法，当 CPU AMX 可用时调用 torch.ops.sgl_kernel.gemma4_rmsnorm_cpu，否则回退到原生实现。
4. 测试配套：更新 test/srt/cpu/test_norm.py，使用 @parametrize 装饰器重构测试函数（如 test_norm、test_l2norm），并新增 test_norm_3d 以验证 3D 输入支持。
5. 扩展注册与契约：在 sgl-kernel/csrc/cpu/torch_extension_cpu.cpp 中注册 gemma4_rmsnorm_cpu 内核，并在 python/sglang/srt/model_executor/cpu_graph_runner.py 中添加相应条目以确保数据契约一致。

关键文件：

- sgl-kernel/csrc/cpu/norm.cpp（模块 内核层；类别 source；类型 core-logic；符号 l2norm_kernel_impl, rmsnorm_kernel_impl）：核心内核实现文件，修改了

l2norm_kernel_impl 和 rmsnorm_kernel_impl 以支持 3D 输入，并添加了 gemma4_rmsnorm_cpu 内核逻辑。

- python/sglang/srt/layers/layernorm.py（模块 模型层；类别 source；类型 core-logic；符号 forward_cpu）：Python 层实现，新增 forward_cpu 方法以集成新内核，是模型前向传播的入口点。
- test/srt/cpu/test_norm.py（模块 归一化测试；类别 test；类型 test-coverage；符号 _norm_test, test_norm, _l2norm_test, test_norm_3d）：测试文件，重构测试并新增 3D 输入覆盖，确保新内核和扩展功能的正确性。
- sgl-kernel/csrc/cpu/torch_extension_cpu.cpp（模块 内核注册；类别 source；类型 core-logic）：Torch 扩展注册文件，添加 gemma4_rmsnorm_cpu 内核的声明和注册，确保在 Python 中可调用。
- python/sglang/srt/model_executor/cpu_graph_runner.py（模块 执行器；类别 source；类型 data-contract）：数据契约文件，添加 gemma4_rmsnorm_cpu 到注册列表，确保在模型执行图中正确识别。

关键符号：l2norm_kernel_impl, rmsnorm_kernel_impl, forward_cpu, gemma4_rmsnorm_cpu, test_norm, test_norm_3d

关键源码片段

sgl-kernel/csrc/cpu/norm.cpp

核心内核实现文件，修改了 l2norm_kernel_impl 和 rmsnorm_kernel_impl 以支持 3D 输入，并添加了 gemma4_rmsnorm_cpu 内核逻辑。

```
template <typename scalar_t>
void l2norm_kernel_impl(
    scalar_t* __restrict__ output,
    const scalar_t* __restrict__ input,
    int64_t batch_size,
    int64_t seq_len, // 新增：支持序列长度维度
    int64_t hidden_size,
    int64_t input_strideB, // 新增：batch维度的步幅
    int64_t input_strideS, // 新增：seq维度的步幅
    int64_t output_strideB,
    int64_t output_strideS,
    float eps = 1e-5) {
  using bVec = at::vec::Vectorized<scalar_t>;
  using fVec = at::vec::Vectorized<float>;
  constexpr int kVecSize = bVec::size();
  // 并行处理batch_size * seq_len个元素，支持3D输入
  at::parallel_for(0, batch_size * seq_len, 0, [&](int64_t begin, int64_t end) {
    int64_t bi{0}, si{0};
    data_index_init(begin, bi, batch_size, si, seq_len); // 初始化索引以处理多维度
    for (int64_t i = begin; i < end; ++i) {
      // 基于batch和seq索引计算指针，支持非连续内存布局
      scalar_t* __restrict__ out_ptr = output + bi * output_strideB + si * output_strideS;
      const scalar_t* __restrict__ input_ptr = input + bi * input_strideB + si * input_strideS;
```

```
      fVec sum_fvec = fVec(float(0));
      float sum_val = float(0);
      int64_t d;
#pragma GCC unroll 4
      for (d = 0; d <= hidden_size - kVecSize; d += kVecSize) {
        bVec x_bvec = bVec::loadu(input_ptr + d);
        fVec x_fvec0, x_fvec1;
        std::tie(x_fvec0, x_fvec1) = at::vec::convert_to_float(x_bvec);
        sum_fvec += x_fvec0 * x_fvec0; // 累加平方和用于方差计算
        sum_fvec += x_fvec1 * x_fvec1;
      }
#pragma GCC unroll 4
      for (; d < hidden_size; ++d) {
        float x_val = static_cast<float>(input_ptr[d]);
        sum_val += x_val * x_val; // 处理剩余元素
      }
      sum_val += vec_reduce_sum(sum_fvec);
      float rsqrt_var = float(1) / std::sqrt(sum_val / hidden_size + eps); // 计算归一化系数
      const fVec scale_fvec = fVec(rsqrt_var);
#pragma GCC unroll 4
      for (d = 0; d <= hidden_size - kVecSize; d += kVecSize) {
        bVec x_bvec = bVec::loadu(input_ptr + d);
        fVec x_fvec0, x_fvec1;
        std::tie(x_fvec0, x_fvec1) = at::vec::convert_to_float(x_bvec);
        x_fvec0 = x_fvec0 * scale_fvec; // 应用归一化
        x_fvec1 = x_fvec1 * scale_fvec;
        bVec out_bvec = convert_from_float_ext<scalar_t>(x_fvec0, x_fvec1);
        out_bvec.store(out_ptr + d); // 存储结果
      }
#pragma GCC unroll 4
      for (; d < hidden_size; ++d) {
        float x_val = static_cast<float>(input_ptr[d]);
        out_ptr[d] = static_cast<scalar_t>(x_val * rsqrt_var);
      }
      data_index_step(bi, batch_size, si, seq_len); // 移动到下一个索引
    }
  });
}
```

python/sglang/srt/layers/layernorm.py

Python 层实现，新增 forward_cpu 方法以集成新内核，是模型前向传播的入口点。

```
def forward_cpu(self, x: torch.Tensor) -> torch.Tensor:
    if _is_cpu_amx_available: # 检查CPU AMX（高级矩阵扩展）是否可用
        # 直接调用新注册的CPU内核，传递权重、eps、scale_shift和with_scale参数
        return torch.ops.sgl_kernel.gemma4_rmsnorm_cpu(
            x, self.weight.data, self.eps, self.scale_shift, self.with_scale
        )
    # 如果AMX不可用，回退到原生实现以确保兼容性
```

```
        return self.forward_native(x)
```

评论区精华

review 中，mingfeima 对 forward_cpu 方法提出修改建议：初始版本检查 x.stride(-1) == 1，但讨论后决定简化处理，直接调用内核，如果遇到非连续输入则报错，避免额外的条件分支。这体现了性能优先的设计权衡，减少了运行时开销。

- forward_cpu 方法中的条件检查简化 (design): 采纳建议，简化实现：仅检查 _is_cpu_amx_available，直接调用内核，不处理非连续输入的特殊情况。

风险与影响

- 风险：技术风险：1. 新内核 gemma4_rmsnorm_cpu 可能引入正确性问题，尤其是对于 3D 输入的处理逻辑；2. 修改现有 l2norm_kernel_impl 和 rmsnorm_kernel_impl 可能影响现有 2D 输入场景的性能或行为，因添加了额外维度参数；3. 依赖 CPU AMX 可用性，在不支持的环境下回退到原生实现，可能导致性能下降。具体风险点在 norm.cpp 的向量化循环和 stride 计算中，需确保边界条件正确。
- 影响：影响范围：对用户，在 CPU 上运行 Gemma 4 模型时将使用优化内核提升推理性能；对系统，修复 CI 失败确保持续集成稳定性，并扩展了归一化内核的通用性；对团队，新增内核需要维护，但测试覆盖充分。影响程度中等，主要影响 CPU 推理路径和 CI 可靠性。
- 风险标记：核心路径变更，新增内核未覆盖所有异常输入

关联脉络

- 暂无明显关联 PR