PR #16775 完整报告

sgl-project/sglang

[CPU] Add GPT-OSS model optimization for CPU

合并时间：2026-05-29 16:05

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/16775

执行摘要

- 一句话：为 CPU 添加 GPT-OSS 模型优化支持
- 推荐动作：值得精读，尤其是 MoE kernel 中 bias 和 swiglu 融合的设计方式、flash_attn 逐行处理以支持 sliding window 的取舍，以及测试重构（@parametrize）模式。建议关注 MXFP4 路径的后续性能基准测试。

功能与动机

本 PR（与 @jianan-gu 合作）为 CPU 添加 GPT-OSS 系列优化支持，包括：BF16 MoE kernel 支持 bias 和 swiglu（移植自 #12537），MXFP4 MoE kernel（依赖 #14385），带 sink 和 sliding window 的注意力（移植自 #12579，依赖 #8666），以及 TP padding 支持（移植自 #12539）。目标是在 CPU 上高效运行 GPT-OSS 架构模型。

实现拆解

1. 扩展 kernel 枚举和接口：在 sgl-kernel/csrc/cpu/gemm.h 中添加 CPUActMethod 枚举（swiglu、silu_and_mul、gelu_and_mul），扩展 CPUQuantMethod 增加 MXFP4 类型；修改 fused_experts 相关函数模板，新增 w1_bias、w2_bias、alpha、limit、act_func、with_bias 等参数，使 kernel 支持多种激活函数和 bias 融合。
2. 实现带 bias 和 swiglu 的 MoE kernel：在 sgl-kernel/csrc/cpu/moe.cpp 中添加 clamp_sigmoid_and_mul 函数，实现 GPT-OSS 的激活函数（带 clamp 和 sigmoid 门控），并重写 fused_experts_kernel_impl，在 stage1 和 stage2 中根据 with_bias 标志选择是否应用 bias，根据 act_func 选择激活函数（silu_and_mul、swiglu 等）。
3. 添加 MXFP4 MoE 量化支持：在 sgl-kernel/csrc/cpu/moe_fp8.cpp 中通过模板 fused_experts_fp_kernel_impl 泛化支持 MXFP4 和 FP8，在 gemm.h 中增加 tinygemm_kernel 的 MXFP4 重载，并在 python/sglang/srt/layers/quantization/mxfp4.py 中集成 Python 调用接口。
4. 实现带 sink 和 sliding window 的注意力：在 sgl-kernel/csrc/cpu/flash_attn.h 中将 flash_attn_softmax 的 apply 方法改为逐行处理（原内部循环移至调用方），以便在 extend 阶段按行应用 sliding window 掩码；对应修改 sgl-kernel/csrc/cpu/extend.cpp 和 decode.cpp 中的调用逻辑。
5. 添加 TP padding 支持：在 python/sglang/srt/models/gpt_oss.py 的 _load_normal_weights 中，当 CPU 且最后一 shard 超出加载权重范围时，自动填充零以

实现 TP 维度对齐。

6. 测试与重构：将 test_moe.py、test_gemm.py 等测试从手动迭代改为 @parametrize 装饰器，新增 test_bf16_moe_bias（测试带 bias 的 MoE）、MXFP4 相关测试，并在 test/registered/cpu/utils.py 中添加 torch_naive_gptoss_fused_moe、moe_gptoss_act、MXFP4QuantizeUtil 等参考实现和工具类。

关键文件：

- sgl-kernel/csrc/cpu/moe.cpp（模块 MoE 内核；类别 source；类型 core-logic；符号 clamp_sigmoid_and_mul, fused_experts_kernel_impl）：核心 MoE kernel 实现，新增 clamp_sigmoid_and_mul 函数，修改 fused_experts_kernel_impl 以支持 bias 和 swiglu 激活。
- sgl-kernel/csrc/cpu/flash_attn.h（模块 注意力内核；类别 source；类型 core-logic；符号 flash_attn_softmax::apply）：注意力软 max 实现修改，将循环外推到调用方，以支持逐行 sliding window 掩码。
- test/registered/cpu/test_moe.py（模块 MoE 测试；类别 test；类型 test-coverage；符号 test_bf16_moe, test_bf16_moe_bias, test_int8_moe, test_fp8_moe）：测试文件，新增 GPT-OSS MoE bias 测试，并重构为 @parametrize 风格。

关键符号：clamp_sigmoid_and_mul, fused_experts_kernel_impl, fused_experts_fp_kernel_impl, flash_attn_softmax::apply, _load_normal_weights (gpt_oss.py)

关键源码片段

sgl-kernel/csrc/cpu/moe.cpp

核心 MoE kernel 实现，新增 clamp_sigmoid_and_mul 函数，修改 fused_experts_kernel_impl 以支持 bias 和 swiglu 激活。

```
// 新增的 clamp_sigmoid_and_mul 函数：实现 GPT-OSS 的激活函数
// 输入为 gate 与 linear 交错的 float 向量，输出为 bf16 结果
// 包含 clamp、sigmoid*gate、(linear+1) 乘法

template <typename scalar_t, int BLOCK_N>
inline void clamp_sigmoid_and_mul(
    scalar_t* __restrict__ output, // 输出 [m_size, N/2] 因为融合后减半
    const float* __restrict__ input0, // 输入来自 gemm 的 float 结果
    int64_t m_size, // 行数
    int64_t N, // 原始 BLOCK_N
    const float alpha, // sigmoid 缩放系数，典型 1.702
    const float limit, // clamp 上限，典型 7.0
    int64_t offset) { // 输出偏移，用于分块存储
  using bVec = at::vec::Vectorized<scalar_t>;
  using fVec = at::vec::Vectorized<float>;
  const fVec one = fVec(1.f);
  const fVec zero = fVec(0.f);
  const fVec limit_v = fVec(limit);
  const fVec nlimit_v = fVec(-limit);
```

```
    const fVec alpha_v = fVec(alpha);

    for (int64_t m = 0; m < m_size; ++m) {
      scalar_t* __restrict__ out = output + m * N;
      const float* __restrict__ cur_ptr = input0 + m * BLOCK_N;
      for (int64_t d = 0; d < BLOCK_N; d += bVec::size()) {
        float tmp_glu0[fVec::size()];
        float tmp_linear0[fVec::size()];
        // 将交错的 gate(x[2i]) 和 linear(x[2i+1]) 分离
        for (int j = 0; j < fVec::size(); ++j) {
          tmp_glu0[j] = cur_ptr[d + j * 2];
          tmp_linear0[j] = cur_ptr[d + j * 2 + 1];
        }
        fVec x0 = fVec::loadu(tmp_glu0);
        fVec y0 = fVec::loadu(tmp_linear0);
        // clamp gate 和 linear
        x0 = at::vec::minimum(x0, limit_v);
        y0 = at::vec::minimum(limit_v, at::vec::maximum(nlimit_v, y0));
        // gate * sigmoid(alpha * gate)
        x0 = x0 / (one + (x0 * alpha_v).neg().exp_u20());
        // (linear + 1) * gate
        y0 = y0 + one;
        x0 = x0 * y0;
        // 以 bf16 存储结果
        convert_from_float_and_store<scalar_t>(out + d / 2 + offset, x0);
      }
    }
  }
```

sgl-kernel/csrc/cpu/flash_attn.h

注意力软 max 实现修改，将循环外推到调用方，以支持逐行 sliding window 掩码。

```
  // flash_attn_softmax 现在只处理单行，不再包含循环
  // 循环移至调用方，以支持 sliding window 时按行设置掩码

  template <typename scalar_t, int BLOCK_M, int BLOCK_N>
  struct flash_attn_softmax {
    static inline void apply(
        float* __restrict__ s_i,
        float* __restrict__ s_delta,
        float* __restrict__ v_prime,
        float* __restrict__ s_prime,
        float* __restrict__ m_prime,
        int64_t m_size, // 实际并未使用，但保留接口
        int64_t n_size,
        int64_t padded_n_size,
        int64_t head_size_v,
        const float sm_scale,
        int64_t row) { // 新增参数：当前处理的行号
```

```
    using Vec = at::vec::Vectorized<float>;
    const Vec scale_vec = Vec(sm_scale);
    float* s_delta = s_i; // 就地计算

    // 当前行的基址
    float* s_i_row = s_i + row * BLOCK_N;
    // 缩放
    at::vec::map<float>([scale_vec](Vec x) { return x * scale_vec; },
                        s_i_row, s_i_row, n_size);
    // 计算最大值
    float m_i = at::vec::reduce_all<float>(
        [](Vec& x, Vec& y) { return at::vec::maximum(x, y); },
        s_i_row, n_size);
    m_i = std::max(m_i, m_prime[row]);
    // ... 后续无改动，省略相同部分
  }
};

// 调用处（在 flash_attn_varlen_kernel_impl 中）
for (int row = 0; row < m_size; ++row) {
    // 可选：在此处设置 sliding window mask（通过修改 s_i 实现）
    flash_attn_softmax<scalar_t, BLOCK_M, BLOCK_N>::apply(
        s_i, s_delta, v_prime, s_prime, m_prime,
        m_size, n_size, padded_n_size, head_size_v, sm_scale, row);
}
```

test/registered/cpu/test_moe.py

测试文件，新增 GPT-OSS MoE bias 测试，并重构为 @parametrize 风格。

```
# 新增的 test_bf16_moe_bias：验证带 bias 的 BF16 MoE 输出
# 使用 @parametrize 简化多参数组合

@parametrize(
    m=[1, 32], n=[128, 64], k=[128, 64], e=[4], topk=[2], renormalize=[False]
)
def test_bf16_moe_bias(self, m, n, k, e, topk, renormalize):
    dtype = torch.bfloat16
    a = torch.randn((m, k), device="cpu", dtype=dtype) / 10
    w1 = torch.randn((e, 2 * n, k), device="cpu", dtype=dtype) / 10
    w1_b = torch.randn((e, 2 * n), device="cpu", dtype=torch.float) / 10 # bias
    w2 = torch.randn((e, k, n), device="cpu", dtype=dtype) / 10
    w2_b = torch.randn((e, k), device="cpu", dtype=torch.float) / 10
    score = torch.randn((m, e), device="cpu", dtype=dtype)
    score = torch.softmax(score, dim=-1, dtype=torch.float32)
    topk_weight, topk_ids = torch.topk(score, topk)
    alpha = 1.702 # GPT-OSS 默认参数
    limit = 7.0
    # 参考实现：PyTorch 原生带 bias 的 MoE
    torch_output = torch_naive_fused_moe_gptoss(
```

```
        a, w1, w2, w1_b, w2_b, topk_weight, topk_ids,
        renormalize, alpha, limit, e
    )
    # 调优后的 fused kernel
    packed_w1 = kernel.convert_weight_packed(w1)
    packed_w2 = kernel.convert_weight_packed(w2)
    fused_output = torch.ops.sgl_kernel.fused_experts_cpu(
        a, packed_w1, packed_w2, topk_weight, topk_ids.to(torch.int),
        False, # inplace 需为 False（见注释）
        CPUQuantMethod.UNQUANT,
        None, None, None, None, None, # scale/zp/block_size
        w1_b, w2_b, alpha, limit, True # bias 和激活参数
    )
    atol = rtol = precision[torch_output.dtype]
    torch.testing.assert_close(torch_output, fused_output, atol=atol, rtol=rtol)
```

评论区精华

- flash_attn_softmax 逐行处理：mingfeima 提问为何要将 softmax 循环从内部移到外部，blzheng 回答需要处理 sliding window 逻辑，因此改为按行处理。
- use_triton_kernels 条件设置：mingfeima 指出 CPU 上原本就会返回 AUTO，不需要额外条件判断，blzheng 同意并恢复。
- shard_dim 设置：mingfeima 发现 shard_dim 已在 L440 设为 -1，后续覆盖为 1 不合理，blzheng 解释为支持 bias 加载而调整。
- TP padding 注释：mingfeima 要求为权重 padding 补零添加注释说明，blzheng 接受。
- 接口简化：mingfeima 指出 w8a8_int8.py 中增加多余参数，需后续简化。
  - flash_attn_softmax 逐行处理原因 (design): 接受该设计，无变更。
  - use_triton_kernels 条件判断简化 (style): 移除多余条件判断。
  - bias 加载时的 shard_dim 设置 (correctness): 经讨论后保留当前实现，后续可能需要进一步清理。
  - TP padding 注释需求 (documentation): blzheng 接受并添加注释。

风险与影响

- 风险：
  - 回归风险：flash_attn_softmax 从批处理改为逐行调用，可能引入单行 softmax 精度差异或影响原先非 sliding window 路径的性能（可接受，因循环外移增加了少量函数调用开销）。
  - 性能风险：MXFP4 MoE kernel 为新增路径，未提供端到端 benchmark 数据，实际推理性能提升待验证。
  - 兼容性风险：kernel 接口增加 w1_bias 等参数，现有调用点（如 w8a8_int8.py）以 None 填充，功能不变，但需确保不破坏非 CPU 后端。
  - 测试覆盖：新增测试集中在单元级别，缺少多节点或大规模场景的集成测试。

- 影响：
  - 用户影响：CPU 用户可运行 GPT-OSS 系列模型（如 DeepSeek-V2）并获得优化后的推理性能，支持 bias 融合和 MXFP4 量化，降低显存和带宽需求。
  - 系统影响：仅影响 CPU 后端，GPU 后端无改动；新增的 CPUActMethod 枚举和 kernel 变体增加了维护成本，但代码结构清晰。
  - 团队影响：需确保 CI 中 CPU 测试覆盖新增路径，MXFP4 依赖的 #14385 需正确合并。
  - 风险标记：核心路径变更，潜在性能回退，新量化路径，测试覆盖待完善

关联脉络

- PR #12537 [CPU] BF16 MoE kernel with bias and swiglu support: 本 PR 移植了该 PR 的 BF16 MoE kernel 修改。
- PR #14385 [CPU] MXFP4 MoE kernel support: 本 PR 依赖该 PR 的 MXFP4 量化支持。
- PR #12579 [CPU] Attention with sink and sliding window: 本 PR 移植了该 PR 的注意力修改（sink/sliding window）。
- PR #8666 [CPU] Attention sink and sliding window base: 本 PR 依赖该 PR 的注意力 sink/sliding window 基础实现。
- PR #12539 [CPU] TP padding support: 本 PR 移植了该 PR 的 TP padding 支持。