PR #16045 完整报告

sgl-project/sglang

[sgl-kernel/cpu] support w8a8 int8 model for arm cpu

合并时间：2026-05-08 14:47

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/16045

执行摘要

- 一句话：支持 Arm CPU W8A8 Int8 模型推理
- 推荐动作：建议精读。该 PR 展示了如何为 sgl-kernel 的 CPU 后端添加新架构支持，包括内核编写、构建系统集成和 Python 路由分层。值得关注的是 i8mm_matmul 的 tile 设计、int8_scaled_mm_with_quant 的量化融合以及 MoE 的完整 int8 流水线。同时 review 中对条件编译的讨论提醒了跨平台代码的兼容性处理技巧。

功能与动机

PR 目的是将 W8A8 Int8 量化模型推理扩展到 ARM CPU 架构，使 ARM 平台也能享受 int8 量化带来的性能优势。PR body 中提供了 GSM8K 精度验证结果，密集模型和 MoE 模型精度与 BF16 持平。

实现拆解

1. 新增 ARM NEON 内核头文件 sgl-kernel/csrc/cpu/aarch64/op.h，定义了基于 sdot 和 i8mm 指令的 tile 矩阵乘法模板 sdot_matmul 和 i8mm_matmul，支持 int8 输入和 bfloat16/float 输出，并集成 per-token/per-channel 缩放因子。
2. 新增 sgl-kernel/csrc/cpu/aarch64/gemm_int8.cpp，实现 int8_scaled_mm_with_quant 函数，先对 bf16 输入做 per-token 量化，再调用 i8mm_matmul 完成 int8 矩阵乘法，输出为 bf16。
3. 新增 sgl-kernel/csrc/cpu/aarch64/moe.cpp，实现 fused_experts_int8_kernel_impl，为 MoE 层做端到端的 int8 前向：复制输入行到连续缓冲区 → 调用 i8mm_matmul 计算 gate 和 up → SiLU 激活 → 计算 down → 加权累加到输出。
4. 修改 python/sglang/srt/layers/quantization/w8a8_int8.py，引入 _is_cpu_arm64 标志，在 process_weights_after_loading 和 apply 方法中添加 ARM 分支，使线性层和 MoE 层使用新的 ARM 内核。
5. 修改构建系统 sgl-kernel/csrc/cpu/CMakeLists.txt，支持按架构子目录 (aarch64/, x86_64/, ppc64/) 选择性编译源文件。
6. 在 torch_extension_cpu.cpp 中用 SGLANG_CPU_ARM64_SKIP_X86_ONLY_OPS 保护 x86 特有操作注册；在 gemm_int8.cpp 中用 #ifndef __aarch64__ 避免 ARM 编译时的重复定义。

关键文件：

- sgl-kernel/csrc/cpu/aarch64/op.h（模块 ARM 内核；类别 source；类型 core-kernel；符号 sdot_matmul, i8mm_matmul, kL1Size, kL2Size）：新增 ARM NEON 内核头文件，定义核心 tile 矩阵乘法模板 sdot_matmul 和 i8mm_matmul，是 ARM int8 计算的基础。
- sgl-kernel/csrc/cpu/aarch64/moe.cpp（模块 ARM 内核；类别 source；类型 core-kernel；符号 fused_experts_int8_kernel_impl）：新增 ARM 融合 MoE 内核，实现 int8 量化的 MoE 前向计算，包括行聚合、gate/up 计算、SiLU 激活、down 计算和加权累加。
- sgl-kernel/csrc/cpu/aarch64/gemm_int8.cpp（模块 ARM 内核；类别 source；类型 core-kernel；符号 int8_scaled_mm_with_quant, int8_scaled_mm_impl）：新增 ARM GEMM 内核入口，提供 int8_scaled_mm_with_quant 函数，将 bf16 输入动态量化为 int8 后调用 i8mm_matmul 计算。
- python/sglang/srt/layers/quantization/w8a8_int8.py（模块 量化层；类别 source；类型 core-logic；符号 _is_cpu_arm64, W8A8Int8LinearMethod.process_weights_after_loading, W8A8Int8LinearMethod.apply, W8A8Int8MoEMethod.process_weights_after_loading）：修改量化层 Python 代码，添加 ARM CPU 分支，使 W8A8Int8 线性层和 MoE 层在新架构上路由到对应内核。
- sgl-kernel/csrc/cpu/CMakeLists.txt（模块 构建系统；类别 infra；类型 infrastructure）：修改构建文件以支持按架构子目录编译，是使 ARM 代码集成进项目的基础。
- sgl-kernel/csrc/cpu/torch_extension_cpu.cpp（模块 扩展注册；类别 source；类型 infrastructure）：修改 torch 扩展注册，用 SGLANG_CPU_ARM64_SKIP_X86_ONLY_OPS 宏保护 x86 特有操作，避免 ARM 构建时链接冲突。
- sgl-kernel/csrc/cpu/gemm_int8.cpp（模块 条件编译；类别 source；类型 infrastructure）：添加 #ifndef __aarch64__保护，避免 ARM 编译时与 aarch64/gemm_int8.cpp 中的函数重复定义。

关键符号：sdot_matmul, i8mm_matmul, fused_experts_int8_kernel_impl, int8_scaled_mm_with_quant, int8_scaled_mm_impl, W8A8Int8LinearMethod.process_weights_after_loading, W8A8Int8LinearMethod.apply, W8A8Int8MoEMethod.process_weights_after_loading, W8A8Int8MoEMethod.apply

关键源码片段

sgl-kernel/csrc/cpu/aarch64/gemm_int8.cpp

新增 ARM GEMM 内核入口，提供 int8_scaled_mm_with_quant 函数，将 bf16 输入动态量化为 int8 后调用 i8mm_matmul 计算。

```
// fused activation quantization and matmul
at::Tensor int8_scaled_mm_with_quant(
    at::Tensor& mat1,
    at::Tensor& mat2,
    at::Tensor& scales2,
    const std::optional<at::Tensor>& bias,
    at::ScalarType out_dtype,
    bool /*is_vnni*/) {
  CHECK_LAST_DIM_CONTIGUOUS_INPUT(mat1);
  CHECK_INPUT(mat2);
```

```
CHECK_INPUT(scales2);
CHECK_DIM(2, mat1);
CHECK_DIM(2, mat2);

int64_t M = mat1.size(0);
int64_t N = mat2.size(0);
int64_t K = mat1.size(1);
int64_t lda = mat1.stride(0);

CHECK_EQ(mat2.size(1), K);
CHECK_EQ(scales2.numel(), N);

const auto st = mat1.scalar_type();
TORCH_CHECK(st == at::kBFloat16, "int8_scaled_mm_with_quant: expect A to be bfloat16.");
TORCH_CHECK(st == out_dtype, "int8_scaled_mm_with_quant: expect A has same dtype with out_dtype.");
TORCH_CHECK(mat2.scalar_type() == at::kChar, "int8_scaled_mm_with_quant: expect mat2 to be int8.");
TORCH_CHECK(scales2.scalar_type() == at::kFloat, "int8_scaled_mm_with_quant: expect scales to be float32.");

const int64_t buffer_size = M * K + M * sizeof(float);
auto buffer = at::empty({buffer_size}, mat1.options().dtype(at::kChar));
auto out = at::empty({M, N}, mat1.options().dtype(out_dtype));

const bool has_bias = bias.has_value();
const float* bias_data = nullptr;
if (has_bias) {
  CHECK_EQ(bias.value().size(0), N);
  bias_data = bias.value().data_ptr<float>();
}

AT_DISPATCH_REDUCED_FLOATING_TYPES(out_dtype, "int8_scaled_mm_with_quant_kernel_impl", [&] {
  int8_t* __restrict__ Aq_data = buffer.data_ptr<int8_t>();
  float* __restrict__ As_data = (float*)((void*)(Aq_data + M * K));
  const scalar_t* __restrict__ A_data = mat1.data_ptr<scalar_t>();

  const int64_t grain = kL1Size / (K * sizeof(scalar_t));
  at::parallel_for(0, M, grain, [&](int64_t begin, int64_t end) {
    for (int64_t m = begin; m < end; ++m) {
      op::quantize_row_int8(Aq_data + m * K, As_data + m, A_data + m * lda, K);
    }
  });

  int8_scaled_mm_impl<scalar_t>(
      out.data_ptr<scalar_t>(),
      Aq_data,
      mat2.data_ptr<int8_t>(),
```

```
        As_data,
        scales2.data_ptr<float>(),
        bias_data,
        M, N, K);
  });
  return out;
}
```

python/sglang/srt/layers/quantization/w8a8_int8.py

修改量化层 Python 代码，添加 ARM CPU 分支，使 W8A8Int8 线性层和 MoE 层在新架构上路由到对应内核。

```
    def process_weights_after_loading(self, layer: torch.nn.Module) -> None:
        if _is_cpu:
            if _is_cpu_amx_available:
                _amx_process_weight_after_loading(layer, ["weight"])
            elif _is_cpu_arm64:
                # Arm64: 保持 weight 原样（不转置），直接标记为不需要梯度
                layer.weight = Parameter(layer.weight.data, requires_grad=False)
            else:
                assert False, "W8A8Int8LinearMethod on CPU only works on AMX or Arm64"
        else:
            # CUDA: 需要转置 weight 以匹配 kernel 期望的列主序
            layer.weight = Parameter(layer.weight.t(), requires_grad=False)
        layer.weight_scale = Parameter(layer.weight_scale.data, requires_grad=False)
```

评论区精华

- alexnails 质疑 ARM 路径是否破坏其他 CPU：cyb70289 回应原 AMX 代码在非 AMX CPU 上同样通过 use_intel_amx_backend 走默认路径，新增的 ARM 分支只是另一个条件分支，不会影响现有逻辑。
- alexnails 担心 gemm_int8.cpp 加入 #ifndef __aarch64__ 造成回归：cyb70289 承认做法粗糙，应保持文件不变纯粹跳过 ARM 构建，并承诺后续提交改进 PR。
- cyb70289 自评 topk_ids = topk_ids.int()：该修正同样适用于原始 AMX 代码，独立于 ARM 支持。
  - ARM 分支修改是否影响其他 CPU (design): cyb70289 解释原 AMX 路径在非 AMX CPU 上已通过 use_intel_amx_backend 绕过，新分支只是增加一个条件，不影响既有逻辑。
  - gemm_int8.cpp 条件编译潜在回归 (design): cyb70289 承认应完全跳过 ARM 构建而非条件编译，承诺后续提交改进 PR。当前做法不会引入功能回归，但代码风格不佳。
  - topk_ids.int() 对 AMX 代码的修复 (correctness): 该修复被确认是必要的，但未追溯应用到 AMX 路径。可能独立修复。

风险与影响

- 风险：

    - 性能风险：ARM 内核尚未经过充分 Benchmark，可能存在性能不及预期的风险。
    - 精度风险：虽然提交者提供了 GSM8K 精度对比，但仅覆盖两个模型，其他模型和框架的兼容性未验证。
    - 条件编译风险：#ifndef __aarch64__ 和 SGLANG_CPU_ARM64_SKIP_X86_ONLY_OPS 的宏逻辑可能遗漏某些 x86 特有代码，导致非 ARM 构建行为异常。
    - 缺失单元测试：当前 PR 未包含针对 ARM 内核的单元测试，回归风险较高。
- 影响：
    - 用户：ARM CPU（如 AWS Graviton、Apple Silicon）用户可运行 W8A8 Int8 模型，显著降低内存占用并提升推理速度。
    - 系统：构建系统增加了架构子目录机制，未来可扩展支持更多 CPU 架构（如 PPC64）。
    - 团队：需要维护 ARM 特定内核，增加后续验证和调试负担。
    - 风险标记：新平台首次支持，缺少单元测试，条件编译风险

关联脉络

- PR #22123 Add Arm64 CPU Phase 1A CI bootstrap: 该 PR 为 Arm64 CPU 添加了 CI 构建与测试基础设施，当前 PR 实现了 Arm64 的推理内核，两者结合使 Arm64 成为完整支持的平台。