

PR #32618 完整报告

sgl-project/sglang

[CPU] Add fp8_per_tensor_scaled_mm_cpu kernel

合并时间: 2026-09-16 12:31

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/32618>

执行摘要

- 一句话: CPU 新增 FP8 per-tensor GEMM 内核, W8A16 量化加速
- 推荐动作: 值得精读。核心亮点是把 per-tensor scale 作为 block-scale 的特例来复用现有 tinygemm 框架, 避免重写一套 GEMM; has_bias 通过模板参数在编译期展开, 配合 AT_DISPATCH_BOOL 避免运行时分支; 测试参数化覆盖了 scale 张量维度、prepack 与 bias 组合。建议读者关注模板泛化对既有调用点的契约影响, 以及 CPU AMX 前置条件在前端如何被强制校验。

功能与动机

PR body 明确指出: 现有 CPU FP8 GEMM 支持 (bmm_cpu 中的 fp8_scaled_mm) 主要面向 batched GEMM, 例如 DeepSeek MLA 路径, 缺少 per-tensor 的 FP8 GEMM 路径, 导致采用 per_tensor 激活方案的 W8A16 模型 (如 mistralai/Ministral-3-14B-Instruct-2512) 无法在 CPU 上使用原生内核。本 PR 的目标是复用已有 CPU FP8 scaled GEMM 实现, 通过把 per-tensor scale 沿 K 分块展开, 补齐该缺失路径。

实现拆解

1. 内核层泛化 (python/sglang/kernels/aot/csrc/cpu/gemm_fp8.cpp): 将 tinygemm_kernel_nn2 从“BF16 x FP8、无 bias”的固定形态扩展为带 packed_t 与 has_bias 模板参数的通用形态; 新增向量化 copy_mul_add_stub (乘 scale 加 bias 的尾段处理); 为 brgemm2 增加 has_bias 特化; tinygemm_kernel2 增加 bias 参数并保持原有 bmm 调用契约。per-tensor scale 的实现方式是把单一标量广播为 vscale, 并在每个 K 分块累加后统一执行 fmadd, 等价于把所有 block scale 视作同一常量。
2. 新增 per-tensor 入口: fp8_per_tensor_scaled_mm_kernel_impl 按 [MB, NB] 二维并行, 使用 AT_DISPATCH_BOOL(bias != nullptr, has_bias, ...) 做编译期分发; 大 M 走 brgemm2, 小 M 走 tinygemm_kernel / tinygemm_kernel2, 并支持 prepack (VNNI) 与未打包权重两种输入。
3. 算子与接口注册: gemm.h 更新 per-tensor 版 tinygemm_kernel 接口 (新增 Bbias 与 do_unpack 参数); torch_extension_cpu.cpp 声明并通过 TORCH_LIBRARY_FRAGMENT 注册 fp8_per_tensor_scaled_mm_cpu 算子; bmm.cpp 为既有 bmm FP8 调用补传 nullptr bias, 避免模板签名变化导致编译失败。
4. 前端接入: cpu_graph_runner.py 为 CPU 图编译注册 fp8_per_tensor_scaled_mm_cpu 的 fake 元信息, 保证 compile 模式可推导输出 shape; fp8.py 的

process_weights_after_loading 增加 CPU AMX 分支（转置权重并做 AMX 预打包后提前 return），apply 中通过 use_intel_amx_backend(layer) 分支调用新内核，并用 output.view(*x.shape[:-1], ...) 还原批次维度。

5. 测试配套：test/registered/cpu/test_gemm.py 新增 test_fp8_per_tensor_gemm，参数化 $M=[1, 11, 97]$ 、 $N=[128, 224]$ 、 $K=[512, 576]$ 、scale_as_vector、has_bias、prepack，以反量化后的 FP32 矩阵乘作为参考进行 assert_close 校验。

关键文件：

- python/sglang/kernels/aot/csrc/cpu/gemm_fp8.cpp（模块 FP8 内核；类别 source；类型 core-logic；符号 fp8_per_tensor_scaled_mm_kernel_impl, tinygemm_kernel_nn2, brgemm2, tinygemm_kernel2）：核心实现文件：泛化 tinygemm_kernel_nn2 / brgemm2 / tinygemm_kernel2 模板，新增 copy_mul_add_stub 与 fp8_per_tensor_scaled_mm_kernel_impl 入口，是 per-tensor FP8 GEMM 的主体。
- python/sglang/srt/layers/quantization/fp8.py（模块 FP8 量化；类别 source；类型 core-logic；符号 Fp8LinearMethod.apply, Fp8LinearMethod.process_weights_after_loading）：前端接入点：process_weights_after_loading 增加 CPU AMX 权重转置与打包分支，apply 中接入 fp8_per_tensor_scaled_mm_cpu 算子，是模型侧启用该内核的关键。
- test/registered/cpu/test_gemm.py（模块 内核测试；类别 test；类型 test-coverage；符号 test_fp8_per_tensor_gemm）：新增 test_fp8_per_tensor_gemm 参数化测试，覆盖 scale 形状、bias、prepack 与多种 M/N/K，是内核正确性的主要保障。
- python/sglang/kernels/aot/csrc/cpu/torch_extension_cpu.cpp（模块 算子注册；类别 source；类型 core-logic；符号 fp8_per_tensor_scaled_mm_cpu）：算子注册层：声明并通过 TORCH_LIBRARY_FRAGMENT 将 fp8_per_tensor_scaled_mm_cpu 注册到 sgl_kernel 命名空间。
- python/sglang/srt/model_executor/cpu_graph_runner.py（模块 图编译；类别 source；类型 data-contract；符号 _）：为 CPU 图编译注册 fp8_per_tensor_scaled_mm_cpu 的 fake 元信息，保证 compile 模式可推导输出 shape。
- python/sglang/kernels/aot/csrc/cpu/gemm.h（模块 内核接口；类别 source；类型 core-logic；符号 tinygemm_kernel）：同步更新 per-tensor 版 tinygemm_kernel 接口，新增 Bbias 与 do_unpack 参数，是模板契约的一部分。
- python/sglang/kernels/aot/csrc/cpu/bmm.cpp（模块 BMM 路径；类别 source；类型 core-logic；符号 bmm_kernel_impl）：适配模板签名变化：既有 bmm FP8 调用补传 nullptr bias，防止编译失败。

关键符号：fp8_per_tensor_scaled_mm_cpu, fp8_per_tensor_scaled_mm_kernel_impl, tinygemm_kernel_nn2::apply, brgemm2::apply, tinygemm_kernel2, copy_mul_add_stub, Fp8LinearMethod.apply, Fp8LinearMethod.process_weights_after_loading, test_fp8_per_tensor_gemm

关键源码片段

[python/sglang/srt/layers/quantization/fp8.py](#)

前端接入点：process_weights_after_loading 增加 CPU AMX 权重转置与打包分支，apply 中接入 fp8_per_tensor_scaled_mm_cpu 算子，是模型侧启用该内核的关键。

CPU + AMX 后端：per-tensor FP8 权重在加载后转置并做 AMX 预打包，
因此在这里提前 return，跳过 Marlin / MXFP8 的后续准备逻辑。

```
if _is_cpu:
    assert _is_cpu_amx_available, (
        "Fp8LinearMethod on CPU requires that CPU has AMX support"
    )
    layer.weight = Parameter(
        layer.weight.data.t().contiguous(), requires_grad=False
    )
    _amx_process_weight_after_loading(layer, ["weight"])
    return
```

... 在 Fp8LinearMethod.apply 中，非 block-quant、非 Marlin、非 MXFP8 且
命中 Intel AMX 后端时，直接调用新增的 per-tensor FP8 GEMM 内核。

```
if use_intel_amx_backend(layer):
    output = torch.ops.sgl_kernel.fp8_per_tensor_scaled_mm_cpu(
        x,
        layer.weight,
        layer.weight_scale,
        bias,
        x.dtype,
        True, # is_vnni: 权重已按 VNNI 格式打包
    )
    # 内核只处理 2D [M, K] x [N, K]^T，这里还原原始批次维度
    return output.view(*x.shape[:-1], layer.weight.shape[0])
```

test/registered/cpu/test_gemm.py

新增 test_fp8_per_tensor_gemm 参数化测试，覆盖 scale 形状、bias、prepack 与多种 M/N/K，是内核正确性的主要保障。

```
@parametrize(
    M=[1, 11, 97],
    N=[128, 224],
    K=[512, 576],
    scale_as_vector=[False, True],
    has_bias=[False, True],
    prepack=[False, True],
)
def test_fp8_per_tensor_gemm(self, M, N, K, scale_as_vector, has_bias, prepack):
    data = torch.randn(M, K, dtype=torch.bfloat16) / 10
    weight = torch.randn(N, K).to(torch.float8_e4m3fn)
    scale = torch.tensor(0.01, dtype=torch.float32)
    # 兼容 scale 为 0 维张量与 1 维向量两种传入方式
    scales = scale.reshape(1) if scale_as_vector else scale
    bias = torch.randn(N, dtype=torch.float32) if has_bias else None

    # 参考实现：反量化 B 到 FP32 后做标准矩阵乘，再乘 per-tensor scale
```

```

ref = torch.matmul(data.float(), weight.float().T) * scale
if bias is not None:
    ref = ref + bias
ref = ref.bfloat16()

kernel_weight = (
    torch.ops.sgl_kernel.convert_weight_packed(weight) if prepack else weight
)
out = torch.ops.sgl_kernel.fp8_per_tensor_scaled_mm_cpu(
    data,
    kernel_weight,
    scales,
    bias,
    data.dtype,
    prepack,
)

atol = rtol = precision[ref.dtype]
torch.testing.assert_close(ref, out, atol=atol, rtol=rtol)

```

评论区精华

review 围绕 bias 的 nullptr 安全与分发方式展开：

- mingfeima 在 `gemm_fp8.cpp` 指出 “bias could be nullptr? so test case did not cover bias is none as well.”，质疑无 bias 路径的测试覆盖。作者 xinguozhu-2026 回复：tiny kernel 基于 `has_bias` 标志访问 bias，真正执行时不会解引用 nullptr，测试已通过 `has_bias=[False, True]` 覆盖两种情况。
- mingfeima 另针对 bias 分发提出 `AT_DISPATCH_BOOL` 建议，作者回复 “done”，最终 `fp8_per_tensor_scaled_mm_kernel_impl` 采用 `AT_DISPATCH_BOOL` 做编译期分支展开，而 `tinygemm_kernel` 封装层用 `if (bias != nullptr)` 选择模板实例，两层配合兼顾可读性与性能。
- bias 为 nullptr 时的内核行为与测试覆盖 (correctness)：确认 `has_bias` 模板分支保证 nullptr 安全，测试两种 bias 场景均已覆盖；后续以 “done” 关闭。
- bias 分发方式：`AT_DISPATCH_BOOL` 与运行期 if 的选择 (design)：采用 `AT_DISPATCH_BOOL` 编译期分发偏置，`has_bias` 分支在编译期展开，避免运行时开销。

风险与影响

- 风险：
 1. 共享内核模板回归面：`gemm_fp8.cpp` 中 `tinygemm_kernel_nn2` / `tinygemm_kernel2` / `brgemm2` 的模板签名变化会影响所有既有调用路径，`bmm.cpp` 已补 nullptr bias 适配，但其他隐藏调用点若未同步更新会编译失败（属于编译期保护，风险可控）。
 2. AMX 硬依赖：`fp8.py` 在 CPU 分支使用 `assert _is_cpu_amx_available`，非 AMX 的 CPU 上启用 W8A16 FP8 会直接报错，需在文档中明确硬件前提。

3. 前端契约隐性: fp8.py 的 apply 依赖 process_weights_after_loading 中 `t().contiguous()` 转置与 AMX 打包的先后顺序, 若其他加载路径绕过该函数, 可能出现 shape 或精度错误; 同时提前 return 会跳过 Marlin/MXFP8 准备逻辑, 约束了 CPU 上的配置组合。
4. 图编译一致性: cpu_graph_runner.py 的 fake 注册只推导 M*N shape, 若算子签名后续演进 (如新增 block_size), 需要同步更新, 否则 compile 模式与 eager 模式行为可能分叉。
5. 精度验证范围有限: PR body 仅给出 gsm8k 200 题结果 (0.935 vs 参考 0.959), 且需临时 PyTorch fallback 量化函数才能启用 Ministral 模型, 正式模型路径的端到端精度覆盖不足。
 - 影响: 对用户: Intel AMX 机器上的 CPU 后端获得原生 per-tensor FP8 GEMM 支持, Ministral-3-14B 等 W8A16 模型不再依赖 bmm 专用路径, 推理精度可达 gsm8k 0.935。对系统: sgl-kernel 的 CPU 算子面新增一个公开算子与一套模板特化, 非 CPU 后端完全不受影响; CPU FP8 线性层从此以 AMX 为硬前提。对团队: sgl-kernel / CPU 维护者需要关注模板泛化对既有 bmm 路径的影响, 并落实 mingfeima 在评论中提醒的 cookbook 更新 (本 PR 未包含文档改动)。
 - 风险标记: 核心内核模板改造, AMX 硬依赖, CPU 量化前端契约变更, bmm 路径回归面

关联脉络

- 暂无明显关联 PR