PR #51583 完整报告

vllm-project/vllm

[CPU] Fold the MXFP4 block scale in 2 instructions instead of 4

合并时间：2026-08-14 16:19

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51583

执行摘要

- 一句话：CPU MXFP4 解包零值特判 4 指令折叠为 2，位级等价
- 推荐动作：值得精读，重点看三处：
  1. vptestmw 互补掩码 + maskz_add 的指令折叠思路——用谓词补集加零掩码写入替代混合选择，是 AVX-512 下常见的两指令等价变换范式。
  2. 测试设计：针对"现有 1e-2 容差会包住坏特判"的盲区，特意构造全零码场景并用精确相等断言，体现了围绕不变量而不是围绕功能正确性写测试的思路。
  3. 作者对性能数据的处理：主动撤回不可复现的测量行、明确区分硬指标（位精确性、指令数）与辅助证据（吞吐），是技术汇报的可借鉴范本。建议关注 sglang#34292 的合入状态，确保两份 vec.h 保持一致。

功能与动机

PR body 明确指出：MXFP4 unpack 把 E8M0 块缩放以整数加法的形式作用到 bf16 指数域，但 E2M1 码本中的两个零码（0x0000 → +0.0、0x8000 → -0.0）没有指数可移，必须特判，原实现写成 and + cmpeq + add + blend 共四条指令；而 vptestmw 恰好给出与旧 cmpeq 互补的谓词（选出非零 lane），配合 maskz_add 的零掩码写入，两条指令即可在每条 lane 得到相同结果。且 vptestmw 是 AVX512BW 指令，vpermw 查找表已需要它，无新增 ISA 要求。作者还强调该路径是 per-byte 工作而非 per-FLOP，在 MXFP4 CPU MoE GEMM 中占有意义比重，值得优化。

实现拆解

1. 变更入口：csrc/cpu/sgl-kernels/vec.h 的 cvt_mxfp4_e2m1_bf16_intrinsic_lut 函数。原实现先 _mm512_setzero_si512 构造 zero 常量，再 _mm512_and_si512 + _mm512_cmp_epi16_mask 检测 (x & 0x7FFF) == 0 的零 lane，随后 _mm512_add_epi16 完成指数加法，最后 _mm512_mask_blend_epi16 把零 lane 混合回 0，每个向量 4 条指令。
2. 核心替换：删除 zero 常量；改用 _mm512_test_epi16_mask(x, abs_mask) 直接获得非零 lane 掩码（与旧掩码互补）；再以 _mm512_maskz_add_epi16 同时完成"非零 lane 加缩放、掩码清掉的 lane 写 0"两件事。每个向量从 4 条指令降到 2 条，共省掉 vpandd ×2、vpcmpeqw ×2、vmovdqu16 ×2，换入 vptestmw ×2，函数整体指令数 16 → 12。该函数服务 CPU MXFP4 MoE GEMM 的 unpack 阶段，属 per-byte 热路径，因此指令削减直接

作用于内核耗时的相关部分。

3. 注释修正（提交 15bb55a）：SGLang 侧 review 指出原注释把新掩码称作“same predicate”具有误导性——新掩码实为旧掩码的补集，结果一致仅因 maskz_add 对掩码清掉的 lane 写 0。作者据此改写注释并同时应用到两仓库拷贝，避免两份 vec.h drift。
4. 测试配套：tests/kernels/moe/test_cpu_quant_fused_moe.py 新增两个参数化测试（共 27 个参数组合）：test_mxfp4_cpu_zero_codes_stay_zero 覆盖 4 个零码字节 × 6 个 E8M0 指数（含 0 与 255 两端），断言 torch.equal 精确为 0，理由是 silu(0) * 0 = 0 会令整层坍缩，任何非零输出都是 unpack 错误而非舍入误差；test_mxfp4_cpu_zero_codes_mixed_with_nonzero 在同一个 32 元素 scale block 内交替零与非零码，强制校验必须是逐 lane 语义而非整块分支，指数限制在 107/127/137 以保证 bf16 可表示（6.0 * 2^(255-127) 会溢出成 inf 与 inf 比较而无意义）。
5. 验证与协同：作者在 2× Xeon Platinum 8592V（Emerald Rapids）、gcc 14.2、torch 2.13.0+cpu 环境下源码构建 CPU 后端，本地跑通 tests/kernels/moe/test_cpu_quant_fused_moe.py 全部 181 个测试，并核验 _C.abi3.so 实际包含 200 个 vptestmw、0 个 vpcmpeqw；穷举对比新旧版本（所有 256 打包字节值 × 256 E8M0 指数共 4,194,304 条 lane、20k 组逐 lane 随机向量共 1,280,000 条 lane、-0.0 码值对全部指数共 16,384 条 lane）均为零差异。同一修改提交至 SGLang（#34292），两仓库可独立合并且顺序无关。

关键文件：

- csrc/cpu/sgl-kernels/vec.h（模块 CPU 内核；类别 source；类型 core-logic；符号 cvt_mxfp4_e2m1_bf16_intrinsic_lut）：主变更文件。cvt_mxfp4_e2m1_bf16_intrinsic_lut 是 MXFP4 CPU MoE GEMM 的 per-byte 解包热路径，zero 特判从 and/cmpeq/add/blend 四指令折叠为 vptestmw + maskz-add 两指令，函数指令数 16→12，无新 ISA 依赖，输出位级等价。
- tests/kernels/moe/test_cpu_quant_fused_moe.py（模块 MoE 测试；类别 test；类型 test-coverage；符号 test_mxfp4_cpu_zero_codes_stay_zero, test_mxfp4_cpu_zero_codes_mixed_with_nonzero）：新增两个针对零码不变量的测试（27 个参数组合）：全零码场景用 torch.equal 断言精确为 0（避免 1e-2 容差掩盖坏特判），混合场景强制逐 lane 语义而非整块分支。

关键符号：cvt_mxfp4_e2m1_bf16_intrinsic_lut, test_mxfp4_cpu_zero_codes_stay_zero, test_mxfp4_cpu_zero_codes_mixed_with_nonzero

关键源码片段

csrc/cpu/sgl-kernels/vec.h

主变更文件。cvt_mxfp4_e2m1_bf16_intrinsic_lut 是 MXFP4 CPU MoE GEMM 的 per-byte 解包热路径，zero 特判从 and/cmpeq/add/blend 四指令折叠为 vptestmw + maskz-add 两指令，函数指令数 16→12，无新 ISA 依赖，输出位级等价。

```
// convert 64 mxfp4 到 2 个 bf16 向量，输入按 32 位方式打包
inline std::tuple<__m512bh, __m512bh> cvt_mxfp4_e2m1_bf16_intrinsic_lut(__m256i a, __m512i
s0, __m512i s1) {
  // LUT：先把 MXFP4 的 16 个 E2M1 码值转成 bf16 表。
  const __m512 values = _mm512_set_ps(MXFP4_VALUES);
```

```
    const __m512i lut = (__m512i)(_mm512_cvtne2ps_pbh(values, values));

    // 只保留低 15 bit（清掉符号位），用于判断是否为 +0.0 / -0.0。
    const __m512i abs_mask = _mm512_set1_epi16(0x7FFF);

    // 把 64 个打包字节展开成 16 位整数，x1 取每个字节的高 nibble。
    __m512i x0 = _mm512_cvtepu8_epi16(a);
    __m512i x1 = _mm512_srli_epi32(x0, 4);

    // 用 LUT 把 mxfp4 的 4 bit 码值映射为 bf16 位模式。
    x0 = _mm512_permutexvar_epi16(x0, lut);
    x1 = _mm512_permutexvar_epi16(x1, lut);

    // 核心优化：vptestmw 对 (x & 0x7FFF) != 0 的 lane 置位，即选中所有非零
    // lane，恰好是旧 and/cmp/add/blend 序列所选零 lane 的补集；配合
    // maskz_add 的零掩码写入，两条指令即可在每条 lane 上与旧序列结果一致。
    // 两个零码 0x0000（+0.0）与 0x8000（-0.0）都被映射为 +0.0，行为与旧
    // 版本完全一致。vptestmw 属 AVX512BW，上面 vpermw 已需要，无新 ISA。
    __mmask32 mask0 = _mm512_test_epi16_mask(x0, abs_mask);
    __mmask32 mask1 = _mm512_test_epi16_mask(x1, abs_mask);

    // 把 E8M0 块缩放当作指数域整数加法执行；掩码未覆盖的 lane 直接写 0。
    x0 = _mm512_maskz_add_epi16(mask0, x0, s0);
    x1 = _mm512_maskz_add_epi16(mask1, x1, s1);

    return std::make_tuple(__m512bh(x0), __m512bh(x1));
  }
```

tests/kernels/moe/test_cpu_quant_fused_moe.py

新增两个针对零码不变量的测试（27 个参数组合）：全零码场景用 torch.equal 断言精确为 0（避免 1e-2 容差掩盖坏特判），混合场景强制逐 lane 语义而非整块分支。

```
  # E2M1 的两个零码：0b0000（+0.0）与 0b1000（-0.0），各在两个 nibble 中。
  MXFP4_ZERO_BYTES = [0x00, 0x88, 0x08, 0x80]
  # 0 与 255 是 E8M0 范围的两端，127 是恒等缩放。
  MXFP4_E8M0_VALUES = [0, 1, 127, 200, 254, 255]


  @pytest.mark.parametrize("zero_byte", MXFP4_ZERO_BYTES)
  @pytest.mark.parametrize("e8m0", MXFP4_E8M0_VALUES)
  def test_mxfp4_cpu_zero_codes_stay_zero(zero_byte, e8m0):
      """零码在任意 E8M0 指数下都必须保持为零。

      unpack 把块缩放实现为 bf16 指数域的整数加法，对 E2M1 码本中除两个
      零码外的所有值都精确；0x0000 与 0x8000 没有指数可移，加上去会变成
      小的有限数而不是 0，因此必须特判。本测试单独钉死这个不变量。

      与 test_mxfp4_cpu_fused_moe 分开的价值：那里零码只是随机权重的一小
      部分，且 1e-2 的容差在低指数下包得住一个坏掉的特判；这里所有权重
```

```
都是零码，输出要么精确为 0，要么就是 unpack 错了。
"""
N, K, E, M = 64, 64, 2, 4
dtype = torch.bfloat16
set_random_seed(0)

a = torch.randn(M, K, dtype=dtype)
w1q = torch.full((E, 2 * N, K // 2), zero_byte, dtype=torch.uint8)
w1s = torch.full((E, 2 * N, K // 32), e8m0, dtype=torch.uint8)
# w2 用普通随机值：零必须穿过第一层 GEMM 与激活存活，非零的 w2
# 才能把"零被破坏"这件事暴露出来。
w2_bf16 = torch.randn(E, K, N, dtype=dtype) / 10
w2q, w2s = MXFP4QuantizeUtil.quantize(w2_bf16)
w2s = w2s.reshape(E, K, N // 32)

topk_weight = torch.ones((M, 1), dtype=torch.float32)
topk_ids = torch.zeros((M, 1), dtype=torch.int32)

pw1, pw1s = _prepack_mxfp4_experts(w1q, w1s)
pw2, pw2s = _prepack_mxfp4_experts(w2q, w2s)
out = ops.fused_experts_cpu(
    a, pw1, pw2, topk_weight, topk_ids, False,
    ops.CPUQuantMethod.MXFP4, pw1s, pw2s, None, None, None,
)

# silu(0) * 0 = 0，整个层坍缩为精确的 0。刻意不用 assert_close：
# 任何非零输出都说明 unpack 错了，而不是舍入误差。
assert torch.equal(out, torch.zeros_like(out)), (
    f"zero code 0x{zero_byte:02x} with e8m0={e8m0} produced "
    f"max |out| = {out.abs().max().item()}"
)
```

评论区精华

评论区最核心的交锋是作者对自己性能数据的主动撤回：

"The number I originally posted was one sample of the fast mode." —— 作者承认最初报告的 unpack_B 循环 1.12x 提升只是快速模式的一次采样，重新测量后多数运行仅 1.006–1.014x，禁用 ASLR 时甚至出现 0.963x。

"At K=3584 that loop reads ~57 kB and writes ~229 kB, so it fits in L2 and is store-bound rather than issue-bound — removing 4 of 16 instructions does not show above the spread." —— 解释了该循环测不出差异的机制原因：数据落在 L2 内且受 store 带宽限制。

"So the claims I would stand behind are the bit-exactness and the instruction count; the throughput number is supporting evidence, not the headline." —— 作者最终把主张收敛为位精确性与指令数两个硬指标，吞吐仅作辅助证据。

- SGLang 侧 review 指出注释中 "same predicate" 表述不准确（实际是补集），作者以 15bb55a 修正注释并同步两份拷贝，保证文件不 drift。
- 红色 CI 检查并非测试失败，而是新贡献者门禁（pre-run-check 需要 verified/ready 标签或 4 个已合并 PR）；作者在本机完整验证后请求维护者加标签，bigPYJ1151 执行 /ci run 触发 Buildkite CI #83834。
- unpack_B 单循环性能数据的撤回 (performance): 作者撤回该行，把主张收敛为位精确性与指令数两个硬指标，端到端 1.15x（out-of-tree 内核）降级为辅助证据。
- 注释把新掩码称为 same predicate 的误导 (design): 作者以 15bb55a 提交修正注释为 complement 并明确写出 maskz_add 行为，同时应用到 vLLM 与 SGLang 两份拷贝避免 drift。
- 新贡献者无法触发 CI 的 pre-run-check 门禁 (other): 维护者 bigPYJ1151 添加标签并执行 /ci run，触发 Buildkite CI #83834。
- 与 SGLang 仓库的同步推进 (other): vLLM 侧已合入；SGLang 侧 PR 仍为 open 状态，需持续跟进确保两份文件一致。

风险与影响

- 风险：
  1. 位级等价依赖穷举验证而非形式证明：穷举覆盖了全部（字节值，E8M0 指数）组合、逐 lane 随机缩放、-0.0 特例，且作者明确覆盖了真实内核中 32 lane 不共享 scale 的 transpose_2x32_16bit 场景，证据充分；但验证在独立编译的二进制上进行，未覆盖真实转置路径的内存布局分支，理论上仍有布局相关差异的残余风险，概率极低。
  2. 性能收益可复现性存疑：作者已撤回 unpack_B 循环的 1.12x 数据；端到端 1.15x 来自 out-of-tree 内核（非 vLLM 官方 MoE 路径），vLLM 自身路径的直接测量缺失，实际收益可能在运行噪声范围内。
  3. 双仓同步风险：csrc/cpu/sgl-kernels/vec.h 源自 SGLang，SGLang 侧 PR（#34292）仍为 open 状态；若一侧长期未合入，未来重新 vendor 该文件会回退此修复，需要两边同时推进。
  4. 兼容性：vptestmw 属 AVX512BW，函数已处于 CPU_CAPABILITY_AVX512 门控内且 vpermw 已需要该 ISA，无新增指令集依赖；非 AVX512 回退路径（pshufb 双表）不受影响。
- 影响：影响范围集中在 CPU 后端 MXFP4 量化 MoE 路径（如 GPT-OSS 类模型）：
  - 用户侧：CPU 部署 MXFP4 量化 MoE 模型可获得无精度代价的潜在吞吐提升（指示性 1.15x），所有输出位模式与旧版本逐位一致，零精度风险；非 AVX512 平台行为不变。
  - 系统侧：AVX-512 内核解包热路径每向量指令数减少 25%（16 → 12），对 per-byte 主导的 unpack 阶段有结构性改善。
  - 团队侧：新增的 27 个参数化零码不变量测试显著抬高未来重构门槛——任何把特判改写为整块分支、或改变掩码语义的改动都会立即失败；测试设计（用 torch.equal 而非 assert_close）本身也是可复用的方法论。
  - 风险标记：位级等价依赖穷举验证，部分性能收益不可复现已撤回，与 SGLang 共享代码需双仓同步，CPU 内核热路径变更

关联脉络

- PR #34292 [CPU] Fold the MXFP4 block scale in 2 instructions instead of 4: SGLang 仓库的对应变更（作者提交），修改同一函数 cvt_mxfp4_e2m1_bf16_intrinsic_lut；vLLM 的 vec.h 源自 SGLang，两仓需保持同步以防未来 re-vendor 回退。