

PR #55352 完整报告

vllm-project/vllm

[CPU] Speedup LM Head on Arm CPUs

合并时间: 2026-09-11 16:21

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/55352>

执行摘要

- 一句话: ACL 下 LM Head 改用 bf16→fp32 内核提速
- 推荐动作: 值得精读, 尤其适合负责 CPU 后端、oneDNN 集成或跨后端性能调优的工程师: 本 PR 用 4 个文件、37 行改动换来 6% e2e 吞吐, 并顺带修正了一个 primitive 缓存键的设计缺陷。重点看三处设计决策: 一是“用问题描述 (problem desc) 而非更换算子”这一最小侵入式提速思路; 二是为什么 c_type 必须进入 class 级缓存键 (reviewer bigPYJ1151 的发现值得反复体会); 三是把 fp32 输出缓冲与一次降采样 copy 放在 PyTorch 侧而非 ACL 内部。阅读时建议同时核对小 M 场景的时间比表与 TODO(fadara01) 热修标记, 评估该阈值在你的流量形态与模型形状下是否成立; 如果后续要把它产品化, 建议把 64000 提为可配置项并清理 CodeRabbit 指出的测试参数拆分问题。

功能与动机

PR body 给出量化数据的量化依据: “LM Head takes around 11-14% of runtime for quantized models”, 并指出根因: “These layers currently fallback to a sub-optimal kernel in ACL. The most optimal kernel only supports bf16 src and wei tensors with fp32 dst.”。因此作者希望“do the oneDNN matmul with bf16:bf16:fp32 problem desc on AArch64 for BF16 layers with N >= 64k”, 并强调“Code path for other backends is not changed”, 最终目标是“This improves throughput of llama-3.1-8b.w8a8 on 32 Neoverse-V3 cores by ~6%”。

实现拆解

1. 变更入口与门控 (csrc/cpu/dnnl_kernels.cpp 的 create_onednn_mm_handler): 在 `#ifdef VLLM_USE_ACL` 内判断 `b.scalar_type() == at::kBFloat16 && b.size(1) >= 64000`, 命中后仅把 `args.c_type` 覆写为 `dnnl::memory::data_type::f32`, 权重类型 `ab_type` 保持 BF16, 从而构造出 bf16 x bf16 -> fp32 问题描述。这样只有大 N 的 BF16 层 (典型 LM head / 大 vocab 输出层) 改变行为, 非 ACL 构建与其它形状完全走原路径。
2. 执行期输出降采样 (onednn_mm): 新增 `dnnl_output` 与 `convert_output` 判断 (handler 输出类型为 f32 且调用方 c 不是 float 时为 true), 为 true 时分配 fp32 临时张量并把 `exec_args.c_ptr` 指向它; `ptr->execute` 之后用 `c.copy_(dnnl_output)` 一次性降回 BF16。ACL 的 bias 退路 (`c = bias; c += matmul(a, b)`) 也改成 `dnnl_output.copy_(bias.value())`, 保证 bias 与 matmul 累加都在 fp32 缓冲里完成、最后统一转换。

3. primitive 缓存与类型暴露 (csrc/cpu/dnnl_helper.h / dnnl_helper.cpp) :
MatMulPrimitiveHandler::ClassMatmulCacheKey 新增 dnnl::memory::data_type c_type 字段, std::hash 特化与 operator== 同步纳入 c_type, get_matmul_cache 构造 class_key 时填入 c_type_; 否则 FP32 输出与 BF16 输出的 matmul 会命中同一个 class 级缓存并复用错误的 primitive。同时在头文件新增 get_output_type() const { return c_type_; } 访问器, 取代早期版本中额外的 bool bf16_in_fp32_out 标志 (该标志在 review 中被指出与基类 Args::c_type 重复后删除)。
4. 测试配套 (tests/kernels/test_onednn.py) : NK_FACTORS 增加 (64128, 4096), 使 test_onednn_gemm 覆盖 $N \geq 64000$ 的新分支; 该参数列表与 test_onednn_int8_scaled_gemm 共用, CodeRabbit 建议拆分但合并时未落实。
5. 演进与遗留: commit 0a2ad60 为初始实现, 9d488e3 按 review 意见“add ctype to primitive cache key, use ctype to determine if output needs downcasting”, c8a2c3a 为 merge main; 源码保留 TODO(fadara01): this is a hot-fix, remove once problem is addressed in the AArch64 stack, 属于明确的临时技术债。

关键文件:

- csrc/cpu/dnnl_kernels.cpp (模块 矩阵乘算子; 类别 source; 类型 core-logic; 符号 create_onednn_mm_handler, onednn_mm) : 变更主体: create_onednn_mm_handler 中的 ACL 门控与 c_type 覆写, 以及 onednn_mm 中的 fp32 临时输出与降采样逻辑都在这里, 直接决定性能收益与正确性。
- csrc/cpu/dnnl_helper.cpp (模块 算子缓存; 类别 source; 类型 core-logic; 符号 std::hash, operator==(MatMulPrimitiveHandler::ClassMatmulCacheKey), MatMulPrimitiveHandler::get_matmul_cache) : primitive 缓存正确性所在: ClassMatmulCacheKey 的 hash / operator== / class_key 构造都必须带上新增的 c_type, 否则 FP32 与 BF16 输出 matmul 会误共享同一 primitive。
- csrc/cpu/dnnl_helper.h (模块 算子缓存; 类别 source; 类型 data-contract; 符号 MatMulPrimitiveHandler::ClassMatmulCacheKey, MatMulPrimitiveHandler::get_output_type) : 类型契约变更: ClassMatmulCacheKey 新增 c_type 字段、MatMulPrimitiveHandler 新增 get_output_type() 访问器, 是 kernels.cpp 判断是否需要降采样的依据; 早期版本的 bool bf16_in_fp32_out 标志在此处被 review 删除。
- tests/kernels/test_onednn.py (模块 算子测试; 类别 test; 类型 test-coverage; 符号 NK_FACTORS, test_onednn_gemm) : 唯一测试改动: NK_FACTORS 增加 (64128, 4096) 以触发 $N \geq 64000$ 的新路径; 同时是 CodeRabbit 未落实建议的落点。

关键符号: create_onednn_mm_handler, onednn_mm, MatMulPrimitiveHandler::get_output_type, MatMulPrimitiveHandler::get_matmul_cache, std::hash, operator==(MatMulPrimitiveHandler::ClassMatmulCacheKey)

关键源码片段

csrc/cpu/dnnl_helper.cpp

primitive 缓存正确性所在：ClassMatmulCacheKey 的 hash / operator== / class_key 构造都必须带上新增的 c_type，否则 FP32 与 BF16 输出 matmul 会误共享同一 primitive。

```
// csrc/cpu/dnnl_helper.cpp
// 对应头文件 csrc/cpu/dnnl_helper.h 中新增的字段：
// struct ClassMatmulCacheKey {
// dnnl_dim_t b_n_size;
// dnnl_dim_t b_k_size;
// dnnl::memory::data_type b_type;
// dnnl::memory::data_type c_type; // 新增：输出类型参与缓存区分
// };

// 哈希函数同步纳入 c_type，否则 FP32 输出与 BF16 输出会落到同一个 bucket
namespace std {
template <>
struct hash<MatMulPrimitiveHandler::ClassMatmulCacheKey> {
    size_t operator()(
        const MatMulPrimitiveHandler::ClassMatmulCacheKey& val) const {
        return hash<dnnl_dim_t>()(val.b_n_size) ^ hash<dnnl_dim_t>()(val.b_k_size) ^
            hash<int>()(static_cast<int>(val.b_type)) ^
            hash<int>()(static_cast<int>(val.c_type));
    }
};
} // namespace std

// 相等比较同样加入 c_type，保证 bf16 输出与 fp32 输出不会复用同一 primitive
bool operator==(const MatMulPrimitiveHandler::ClassMatmulCacheKey& l,
                const MatMulPrimitiveHandler::ClassMatmulCacheKey& r) {
    return l.b_n_size == r.b_n_size && l.b_k_size == r.b_k_size &&
        l.b_type == r.b_type && l.c_type == r.c_type;
}

// 首次查缓存时构造 class key，必须带上 c_type_，与上面的 hash/== 保持一致
// （这是 review 中 bigPYJ1151 指出的问题：FP32 与 BF16 输出会命中同一 cache）
dnnl::matmul MatMulPrimitiveHandler::get_matmul_cache(
    const MSizeCacheKey& key) {
    if (m_size_cache_.get() == nullptr) {
        ClassMatmulCacheKey class_key = {.b_n_size = b_n_size_,
                                           .b_k_size = b_k_size_,
                                           .b_type = b_type_,
                                           .c_type = c_type_};

        m_size_cache_ =
            get_matmul_class_primitive_cache(class_key, primitive_cache_size_);
    }
    return m_size_cache_->get_or_create(key, [&]() {
        dnnl::matmul::primitive_desc desc = this->create_primitive_desc(key, false);
        auto manager = cpu_utils::ScratchPadManager::get_scratchpad_manager();
        manager->realloc(desc.scratchpad_desc().get_size());
        return dnnl::matmul(desc);
    });
}
```

```
});  
}
```

tests/kernels/test_onednn.py

唯一测试改动：NK_FACTORS 增加 (64128, 4096) 以触发 $N \geq 64000$ 的新路径；同时是 CodeRabbit 未落实建议的落点。

```
# tests/kernels/test_onednn.py  
# 新增 (64128, 4096) 让 test_onednn_gemm 走到  $N \geq 64000$  的 bf16:bf16->fp32 新分支;  
# 注意该列表被 test_onednn_gemm 与 test_onednn_int8_scaled_gemm 共用,  
# CodeRabbit 建议拆成 GEMM 专用参数以避免 INT8 用例也分配大张量, 但合并时未落实。  
NK_FACTORS = [  
    (256, 128),  
    (4096, 4096),  
    (16384, 4096),  
    (64128, 4096), # 触发 ACL bf16 x bf16 -> fp32 路径 (LM head 形状)  
    (1023, 491),  
    (1001, 15),  
]
```

评论区精华

1) bigPYJ1151 质疑新增的 `bf16_in_fp32_out` 标志冗余：“I think this is not required as `DNNLMatMulPrimitiveHandler::Args` already contains `c_type`.” 作者 fadara01 回应“I added it to make things more explicit for people reading the code. i'll remove it if you feel strongly against it.”，最终删除并改为暴露 `get_output_type()`，复用基类已有的 `c_type_`。
2) bigPYJ1151 发现缓存键缺陷：“`c_type` needs to be added, otherwise FP32 and BF16 output matmul will match same primitive cache.” 作者承认“this can't happen in practise... but I agree that we should add it to the cache key.”，随后在 commit 9d488e3 中修复，评审质量较高。3) CodeRabbit 对测试参数提出建议：把 (64128, 4096) 从共享 `NK_FACTORS` 移到 GEMM 专用参数列表，理由是 `test_onednn_int8_scaled_gemm` 走 `create_onednn_scaled_mm`、无法覆盖 `bf16_in_fp32_out`，却要为整个 INT8 参数矩阵分配大张量；该建议在合并时仍未落实。4) 门控阈值 $N \geq 64000$ 、TODO 热修标记、以及小 M 场景的时间比 < 1 未在 review 中展开讨论，属于遗留观察点。

- `MatMulPrimitiveHandler::Args` 中 `bf16_in_fp32_out` 标志是否冗余 (design): 删除额外的 `bool` 标志，改为复用基类已保存的 `c_type_` 并新增 `get_output_type()` 访问器，避免同一语义存在两份状态。
- class 级 primitive 缓存键必须区分输出类型 `c_type` (correctness):
`ClassMatmulCacheKey` 新增 `c_type` 字段，`std::hash` 特化、`operator==` 与 `get_matmul_cache` 中的 `class_key` 构造同步更新 (commit 9d488e3)，消除了 FP32/BF16 输出误共享 primitive 的隐患。
- 大 shape 是否应放在共享 `NK_FACTORS` 参数列表中 (testing): 未处理：合并时的 head 版本仍把该形状留在 `NK_FACTORS`，INT8 用例因此也要为新增形状分配张量，测试耗时与内存开销增加。

风险与影响

- 风险：1) 回归风险（小 batch）：PR body 的 old/new 时间比表中，M=1 以及 M=7~8 的部分配置小于 1（如 Llama-3 96 线程 M=7 为 0.898x、Gemma-4 96 线程 M=8 为 0.897x），说明低并发小 batch 下新路径可能轻微变慢，原因是额外分配 fp32 输出并整张量 copy；收益集中在 M >= 32（可达 1.7x~3.0x）。2) 硬编码阈值：b.size(1) >= 64000 是魔法数字（并非 64k=65536），且只看 N 不看 M，N 落在 [64000, 65536) 的层也会进入新路径，未来模型形状变化时缺乏可配置入口（未走 envs.py 或 config）。3) 内存与拷贝开销：convert_output 为 true 时每次调用都会 torch::empty(c.sizes(), at::kFloat)，对 N≈248320、M=256 的 LM head 约为 254 MB 的瞬时 fp32 缓冲，并伴随一次全量 dtype 转换 copy；对 Arm CPU 内存带宽敏感的场景可能有副作用。4) 正确性与数值一致性：新路径在 fp32 中累加后一次性舍入到 BF16，精度通常不劣于原路径，但与旧实现的输出 bit 不完全一致，可能影响依赖严格 logits 比对的测试或用户复现。5) 缓存键变更影响面：ClassMatmulCacheKey 增加 c_type 会改变 class 级 primitive 缓存的分区粒度（DNNLPPrimitiveCache 静态容量 128），仅在 c_type != b_type（即 ACL + 大 N BF16）时多出条目；它同时修复了 FP32/BF16 输出误共享 primitive 的隐患，属于净收益。6) 覆盖范围局限：改动完全位于 #ifdef VLLM_USE_ACL，只有以 ACL 后端构建的 AArch64 包受益，x86 oneDNN 与其它 CPU 构建不受影响也不承担风险；但这也意味着收益依赖构建矩阵，且 TODO 热修需要在上游 AArch64 栈修好后主动清理。
- 影响：用户侧：在 Arm Neoverse/AArch64 + ACL 的 CPU 部署上，BF16 与 W8A8 等量化模型的长 prompt、大 batch prefill/ 解码场景收益最明显（LM head 层在大 M 下可达约 2~3x 加速，e2e 约 6% 吞吐提升）；低并发单流场景基本持平甚至略慢，用户感知取决于流量形态。系统侧：LM Head/logits 阶段占量化模型 11%~14% 运行时，该优化直接压缩这部分占比，同时把一次 FP32->BF16 转换从 ACL 内核内部移到 PyTorch 张量 copy，改变了执行期内存分配模式（每次调用新增 fp32 临时缓冲）。团队侧：代码路径隔离良好、改动面小，但引入了带 TODO 的临时分支与硬编码阈值，需要在 ACL/AArch64 上游修复后回收；同时 class 级 primitive 缓存键语义的变化需要后续修改 MatMulPrimitiveHandler 的同学知晓。
- 风险标记：硬编码阈值 64000，ACL 热修待清理，小 M 场景轻微回退，临时 FP32 缓冲开销，测试建议未落实

关联脉络

- PR #54968 [XPU] Add forward_xpu to Mixer2RMSNormGated and FusedRMSNormGated: 同为非 NVIDIA 硬件后端通过把热点层接线到更优实现来换取端到端性能提升的 PR（XPU eager 提速 25%），与本 PR 在 ACL 上切换到最快 hybrid kernel 的方法论可对比阅读。
- PR #51692 [ROCm][Perf] Add bpreshuffled blockscaled fp8 GEMM: 同样是在特定硬件后端为 GEMM 引入受类型 / 形状条件门控的专用内核，并需要同步调整内核选择与缓存 / 预打包逻辑；kernel 门控条件与缓存键的权衡话题共通。