PR #38445 完整报告

vllm-project/vllm

[PERF]MiniMax-M2 gate kernel

合并时间：2026-05-30 09:28

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/38445

执行摘要

- 一句话：融合 MiniMax-M2 MoE 门控的 FP32 路由 GEMM 核函数
- 推荐动作：值得精读，展示了如何为特定模型定制融合 GEMM 并通过分层调度集成到现有 MoE 门控框架。重点可关注 GateLinear.forward 的四级调度设计和 fp32_router_gemm_fake 的注册模式。

功能与动机

MiniMax-M2 的 MoE 门控需要以 FP32 计算，原始实现需要先转换为 fp32 再调用 cuBLAS GEMM，导致最多 3 个内核启动。融合 kernel 可以减少内核启动开销，在低并发交互场景下获得显著性能提升。

实现拆解

1. 新增 CUDA 核函数（csrc/libtorch_stable/fp32_router_gemm.cu）：针对 MiniMax-M2 的固定维度（H=3072, E=256, M<=32）实现融合 FP32 GEMM，支持 bf16 和 fp32 激活输入，权重始终为 fp32。
2. 注册自定义操作（vllm/_custom_ops.py）：添加 fp32_router_gemm 高层封装及 fp32_router_gemm_fake 用于 torch.compile 图捕获。
3. 扩展现有调度（vllm/model_executor/layers/fused_moe/router/gate_linear.py）：在 GateLinear.forward 中新增第二级 fp32 专用 kernel 调度，并加入 eligibility 检查（SM90+、非 bias、精确维度、batch≤32）。
4. 适配模型（vllm/model_executor/models/minimax_m2.py）：将门控层从 ReplicatedLinear 替换为 GateLinear，移除手动 hidden_states.to(torch.float32) 转换。
5. 构建配置（CMakeLists.txt、cmake/utils.cmake）：集成新 CUDA 源码并添加 cuda_archs_sm90plus 函数简化架构检测。
6. 基准与测试：新增 benchmarks/kernels/benchmark_router_gemm.py 覆盖性能对比，tests/kernels/test_fp32_router_gemm.py 覆盖正确性验证。

关键文件：

- vllm/model_executor/layers/fused_moe/router/gate_linear.py（模块 MoE 路由；类别 source；类型 core-logic；符号 fp32_router_gemm_dispatch_impl, fp32_router_gemm_dispatch_fake, FP32_SUPPORTED_NUM_EXPERTS, FP32_SUPPORTED_HIDDEN_SIZES）：核心调度层，新增 fp32 专用 kernel 的

eligibility 检查与 forward 中 Tier 2 分支

- vllm/_custom_ops.py（模块 自定义操作；类别 source；类型 core-logic；符号 fp32_router_gemm, fp32_router_gemm_fake）：注册 fp32_router_gemm 自定义操作及其 fake 实现，作为 Python 与 CUDA kernel 的桥梁
- benchmarks/kernels/benchmark_router_gemm.py（模块 基准测试；类别 source；类型 entrypoint；符号 FP32_SUPPORTED_NUM_EXPERTS, FP32_SUPPORTED_HIDDEN_SIZES, FP32_MAX_TOKENS, get_benchmark）：新增性能基准脚本，涵盖 fp32 kernel 与其他路由 GEMM 的对比
- tests/kernels/test_fp32_router_gemm.py（模块 核函数测试；类别 test；类型 test-coverage；符号 _requires_sm90, _ref, test_fp32_activation, test_bf16_activation）：验证 fp32 router gemm 输出形状、dtype 及数值精度
- csrc/libtorch_stable/fp32_router_gemm.cu（模块 CUDA 核；类别 other；类型 entrypoint）：融合 GEMM 的 CUDA 核函数实现，核心性能所在
- csrc/libtorch_stable/fp32_router_gemm_entry.cu（模块 CUDA 入口；类别 other；类型 entrypoint）：CUDA kernel 的 torch 绑定入口，实现算子注册
- vllm/model_executor/models/minimax_m2.py（模块 模型适配；类别 source；类型 data-contract；符号 GateLinear）：模型入口，将门控层替换为 GateLinear 并移除手动 fp32 转换
- cmake/utils.cmake（模块 构建配置；类别 infra；类型 configuration；符号 cuda_archs_sm90plus）：添加 cuda_archs_sm90plus 函数，简化 SM90+ 架构检测
- CMakeLists.txt（模块 构建配置；类别 infra；类型 configuration）：集成 fp32_router_gemm 源码至构建系统

关键符号：fp32_router_gemm, fp32_router_gemm_fake, GateLinear.forward, fp32_router_gemm_dispatch_impl, fp32_router_gemm_dispatch_fake, get_benchmark

关键源码片段

vllm/model_executor/layers/fused_moe/router/gate_linear.py

核心调度层，新增 fp32 专用 kernel 的 eligibility 检查与 forward 中 Tier 2 分支

```
# file: vllm/model_executor/layers/fused_moe/router/gate_linear.py (partial)

class GateLinear(ReplicatedLinear):
    # FP32 专用 kernel 的维度约束
    FP32_SUPPORTED_NUM_EXPERTS = [256]
    FP32_SUPPORTED_HIDDEN_SIZES = [3072]
    FP32_MAX_TOKENS = 32

    def forward(self, x: torch.Tensor) -> torch.Tensor | tuple[torch.Tensor, Parameter | None]:
        # Tier 1: DSV3 专用 kernel (DeepSeek V3)
        if self.allow_dsv3_router_gemm and x.shape[0] <= 16:
            output = ops.dsv3_router_gemm(
                hidden_states=x,
                router_weight=self.weight,
```

```
                output_dtype=torch.float32,
            )
            return output, None

        # Tier 2: FP32 专用 kernel (MiniMax-M2, H=3072, E=256, M<=32)
        # 使用自定义 op 使得 torch.compile 不会冻结运行时 token 数分支
        if self.allow_fp32_router_gemm and x.dtype in (torch.float32, torch.bfloat16):
            output = torch.ops.vllm.fp32_router_gemm_dispatch(x, self.weight)
            return output, None

        # Tier 3: cuBLAS bf16×bf16→fp32 (SM90+ 且权重是 bf16)
        if self.allow_cublas_router_gemm and x.dtype == torch.bfloat16:
            output = torch.ops.vllm.cublas_router_gemm(x, self.weight)
            return output, None

        # Tier 4: 降级到基类 F.linear
        return super().forward(x)
```

vllm/_custom_ops.py

注册 fp32_router_gemm 自定义操作及其 fake 实现，作为 Python 与 CUDA kernel 的桥梁

```
# file: vllm/_custom_ops.py (partial)

def fp32_router_gemm(
    hidden_states: torch.Tensor,
    router_weight: torch.Tensor,
) -> torch.Tensor:
    # 分配输出：shape (batch_size, num_experts)，始终为 fp32
    output = torch.empty(
        hidden_states.shape[0],
        router_weight.shape[0],
        device=hidden_states.device,
        dtype=torch.float32,
    )
    # 调用底层 CUDA kernel
    torch.ops._C.fp32_router_gemm(output, hidden_states, router_weight)
    return output

# 注册 fake op 用于 torch.compile 的图捕获
if hasattr(torch.ops, "_C") and hasattr(torch.ops._C, "fp32_router_gemm"):

    @register_fake("_C::fp32_router_gemm")
    def fp32_router_gemm_fake(
        output: torch.Tensor,
        mat_a: torch.Tensor,
        mat_b: torch.Tensor,
    ) -> None:
        return
```

benchmarks/kernels/benchmark_router_gemm.py

新增性能基准脚本，涵盖 fp32 kernel 与其他路由 GEMM 的对比

```
# file: benchmarks/kernels/benchmark_router_gemm.py (partial)

# fp32 专用 kernel 的支持维度
FP32_SUPPORTED_NUM_EXPERTS = [256]
FP32_SUPPORTED_HIDDEN_SIZES = [3072]
FP32_MAX_TOKENS = 32

def get_benchmark(model, max_batch_size, trust_remote_code):
    @triton.testing.perf_report(...)
    def benchmark(batch_size, provider):
        # ... 解析配置 ...
        is_fp32_router_model = (
            is_hopper_or_blackwell
            and num_experts in FP32_SUPPORTED_NUM_EXPERTS
            and hidden_size in FP32_SUPPORTED_HIDDEN_SIZES
        )
        allow_fp32_router_gemm = is_fp32_router_model and batch_size <= FP32_MAX_TOKENS

        if provider == "torch":
            def runner():
                if allow_fp32_router_gemm:
                    F.linear(mat_a.float(), mat_b) # torch 侧回退到手动 fp32 转换
                elif has_bias:
                    F.linear(mat_a, mat_b, bias)
                else:
                    F.linear(mat_a, mat_b)
        elif provider == "vllm":
            def runner():
                if allow_dsv3_router_gemm:
                    ops.dsv3_router_gemm(mat_a, mat_b, torch.bfloat16)
                elif allow_fp32_router_gemm:
                    ops.fp32_router_gemm(mat_a, mat_b) # 使用新 kernel
                elif allow_gpt_oss_router_gemm:
                    ops.gpt_oss_router_gemm(mat_a, mat_b, bias)
                elif is_fp32_router_model:
                    F.linear(mat_a.float(), mat_b) # batch>32 时回退
                else:
                    F.linear(mat_a, mat_b)
        # ... 执行基准并计算 TFLOPs ...
```

评论区精华

- bias 防护缺失：claude[bot] 指出 allow_fp32_router_gemm 未检查 not bias，可能导致 bias 被忽略。评论后已添加 not bias 条件。

- CMake 重复逻辑：gemini-code-assist 建议将 SM90+ 架构检测提取为函数。后创建 cuda_archs_sm90plus 并采纳 Harry-Chen 的 SM120 支持建议。
- 冗余 if-pass：minimax_m2.py 中残留无操作 if num_tokens <= 32: pass，已按评论移除。
- Benchmark 回退：claude[bot] 指出 vllm provider 在 batch>32 时因缺少回退而崩溃。最终代码通过 elif is_fp32_router_model: F.linear(...) 处理了该情况。
- 注释错误：claude[bot] 指出 moe_ops.h 注释误写为 "FP16 x FP32" 实际为 BF16，未见明确修复。
  - FP32 kernel 缺少 bias 防护 (correctness): 已添加 not bias 检查
  - CMake 重复逻辑重构 (design): 已创建 cuda_archs_sm90plus 函数并添加 SM120
  - 冗余 if-pass 代码 (style): 已移除
  - Benchmark vllm provider 缺少回退 (correctness): 已添加 elif is_fp32_router_model: F.linear(mat_a.float(), mat_b) 回退路径
  - Kernel 注释错误 (documentation): 未见明确修复，注释仍可能误导
  - SM120 架构支持 (design): 已添加 SM120 (12.0) 到架构列表

风险与影响

- 风险：仅支持 SM90+（Hopper/Blackwell），旧 GPU 回退到通用路径，无风险但无收益。维度硬编码（H=3072, E=256），其他模型不能利用。高并发（64）下效果消失，TTFT 在并发 16 时轻微倒退（+10%），需进一步排查。新 kernel 数值精度与参考路径在 atol=2e-4 (fp32) / 2e-2 (bf16) 内一致，但若未来调整 kernel 可能引入偏差。
- 影响：直接影响 MiniMax-M2 模型在 SM90+ GPU 上的门控性能，对 vLLM 其他模型无影响。低并发（1-32）吞吐提升 6%-32%，TTFT 改善 4%-29%。不改变 API 和协议。新增文件约 600 行 CUDA/C++，对构建时间和二进制大小影响有限。
- 风险标记：SM90+ 依赖，维度固定，高并发退化，数值精度敏感

关联脉络

- 暂无明显关联 PR