PR #31552 完整报告

sgl-project/sglang

[Performance] Speed up Marlin MoE with occupancy-aware launch specialization

合并时间：2026-07-25 19:38

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/31552

执行摘要

- 一句话：Marlin MoE kernel occupancy 调度与编译专门化优化
- 推荐动作：建议技术管理者将此 PR 作为 kernel 性能优化的典型案例阅读，重点关注 determine_exec_config 中 occupancy 计算逻辑以及 if constexpr 与 JIT 编译的结合方式。对 Marlin MoE 路径的其他量化变体（如 NVFP4）也可借鉴类似优化思路。

功能与动机

原始 Marlin MoE kernel 在 hot loop 中通过运行时 bool 分支处理 expert-parallel 和 bias 路径，且 large-M 场景下直接接受第一个有效的 launch 配置而非评估 occupancy。PR body 指出两种变化存在材料交互（"either change alone is small, while the combined kernel consistently improves"），因此需要联合优化以最大化 H200 上的推理性能。

实现拆解

1. 添加编译时常量模板参数：在 marlin_template.h 和 kernel.h 的 Marlin 模板声明中新增 kIsEP 和 kHasBias 两个 bool 模板参数，替代原有的运行时参数 is_ep 和 has_bias。
2. 使用 if constexpr 消除运行时分支：在 kernel 的 block 调度（专家 ID 扫描）和 bias 累加逻辑中，将 if (is_ep) 改为 if constexpr (kIsEP)，将 if (has_bias) 改为 if constexpr (kHasBias)，使得编译器为不同路径生成独立无分支的代码。
3. 优化 launch 配置选择：在 moe_wna16_marlin.cuh 的 determine_exec_config 中，不再接受第一个有效配置，而是通过 cudaFuncGetAttributes 获取 kernel 的寄存器数和共享内存使用量，结合设备总资源和问题并行度（prob_n / th_config.thread_n * ...）计算允许的并发 block 数 allow_count，选取 allow_count 最大的配置。同时引入 kSharedMemoryValidityMargin（512）和 kSharedMemoryLaunchReserve（1024）两个常量以更严格地判断共享内存合法性。
4. 更新 Python JIT 接口：moe_wna16_marlin.py 中 _jit_moe_wna16_marlin_module 新增 is_ep 和 has_bias 参数，并传递给 make_cpp_args，使得不同路径编译为独立的 kernel 实例，避免 if constexpr 退化。
5. 新增测试覆盖 large non-EP 路径：test_moe_wna16_marlin.py 新增 test_fused_marlin_moe_large_non_ep_schedule，参数化 m ∈ {123, 2304} 和 has_bias ∈ {False, True}，对比 fused_marlin_moe 输出与逐 expert 参考实现，容忍度 rtol=0.04 atol=0.06。

关键文件：

- python/sglang/kernels/jit/csrc/gemm/marlin_moe/marlin_template.h（模块 jit-kernel；类别 source；类型 core-logic）：核心 CUDA kernel 模板，添加 kIsEP 和 kHasBias 编译时常量，使用 if constexpr 消除运行时分支，直接影响热循环效率。
- test/registered/kernels/ops/moe/test_moe_wna16_marlin.py（模块 测试；类别 test；类型 test-coverage；符号 test_fused_marlin_moe_large_non_ep_schedule）：新增 test_fused_marlin_moe_large_non_ep_schedule 参数化测试，覆盖 large-M 和 bias 组合，验证数值正确性。
- python/sglang/kernels/jit/csrc/gemm/marlin_moe/moe_wna16_marlin.cuh（模块 jit-kernel；类别 other；类型 core-logic）：实现 occupancy-aware launch 配置选择逻辑的辅助文件，新增 register/shared-memory 计算和允许并发 block 数估算，是性能提升的关键逻辑。

关键符号：get_marlin_kernel, determine_exec_config, is_valid_config, _jit_moe_wna16_marlin_module, moe_wna16_marlin_gemm, test_fused_marlin_moe_large_non_ep_schedule

关键源码片段

python/sglang/kernels/jit/csrc/gemm/marlin_moe/marlin_template.h

核心 CUDA kernel 模板，添加 kIsEP 和 kHasBias 编译时常量，使用 if constexpr 消除运行时分支，直接影响热循环效率。

```
// marlin_template.h（片段）
// 编译时常量模板参数替代运行时 bool，实现零分支专门化

template <
    typename scalar_t, // compute dtype, half or nv_float16
    const host::ScalarTypeId w_type_id, // weight ScalarType id
    const int threads, // number of threads in a threadblock
    const int thread_m_blocks, // number of 16x16 blocks in the m dimension
    const int thread_n_blocks, // same for n dimension (output)
    const int thread_k_blocks, // same for k dimension (reduction)
    const bool m_block_size_8, // whether m_block_size == 8
    const int stages, // number of stages for async pipeline
    const int group_blocks, // number of consecutive 16x16 blocks with separate scale
    const bool is_zp_float, // is zero point of float16 type?
    const bool kIsEP, // 编译时已知：是否 expert-parallel 路径
    const bool kHasBias // 编译时已知：是否有 per-expert bias
    >
__global__ void Marlin(...) {
    // ...
    // 示例：专家 ID 扫描中根据 kIsEP 使用 if constexpr
    int num_valid_blocks = parallel;
    if constexpr (kIsEP) {
        // EP 路径需要过滤无效 block
        for (int i = 0; i < parallel; i++) {
```

```
            if (expert_ids_ptr[i] == -1) num_valid_blocks--;
        }
    }
    // ...
    // bias 累加中根据 kHasBias 使用 if constexpr
    if constexpr (kHasBias) {
        if (last) {
            scalar_t2 tmp_bias = b_bias[0];
            if constexpr (m_block_size_8) {
                tmp_bias = Dtype::num2num2(
                    reinterpret_cast<scalar_t*>(&b_bias[0])[(threadIdx.x % 8) / 4]);
            }
            res = __hadd2(res, tmp_bias);
        }
    }
    // ...
}
```

test/registered/kernels/ops/moe/test_moe_wna16_marlin.py

新增 test_fused_marlin_moe_large_non_ep_schedule 参数化测试，覆盖 large-M 和 bias 组合，验证数值正确性。

```
# test_moe_wna16_marlin.py（新增测试）
# 参数化验证 large-M 和 bias 组合下的 Marlin MoE 数值正确性

@pytest.mark.parametrize("m", [123, 2304])
@pytest.mark.parametrize("has_bias", [False, True])
def test_fused_marlin_moe_large_non_ep_schedule(m, has_bias):
    torch.manual_seed(0)
    n, k, e, topk = 1024, 512, 8, 2
    dtype = torch.bfloat16
    group_size = 128
    quant_type = scalar_types.uint4b8

    # 随机初始化输入和量化权重
    hidden_states = torch.randn((m, k), device="cuda", dtype=dtype) / 10
    w_ref1, qweight1, scales1, zeros1, g_idx1, sort_indices1 = _setup_moe_weights(
        e, n, k, quant_type, group_size, False, dtype
    )
    w1_bias = (
        torch.randn((e, n), device="cuda", dtype=dtype) / 100 if has_bias else None
    )
    # ... 第二个权重类似初始化 ...

    # 调用 fused_marlin_moe（与 JIT 编译接口对齐）
    output = fused_marlin_moe(
        hidden_states=hidden_states,
        w1=qweight1, w2=qweight2,
        w1_scale=scales1, w2_scale=scales2,
```

```
        gating_output=router_logits,
        topk_weights=topk_weights, topk_ids=topk_ids,
        g_idx1=g_idx1, g_idx2=g_idx2,
        sort_indices1=sort_indices1, sort_indices2=sort_indices2,
        w1_zeros=zeros1, w2_zeros=zeros2,
        w1_bias=w1_bias, w2_bias=w2_bias,
        num_bits=4, is_k_full=True,
        routed_scaling_factor=1.0, activation="relu2", is_gated=False,
    )

    # 逐 expert 参考实现（含 bias 分支）
    output_ref = torch.zeros_like(hidden_states, dtype=torch.float32)
    for expert_id in range(e):
        token_indices, route_indices = torch.where(topk_ids == expert_id)
        intermediate = hidden_states[token_indices] @ w_ref1[expert_id].T
        if w1_bias is not None:
            intermediate += w1_bias[expert_id]
        intermediate = torch.square(torch.relu(intermediate))
        routed = intermediate @ w_ref2[expert_id].T
        if w2_bias is not None:
            routed += w2_bias[expert_id]
        output_ref.index_add_(
            0, token_indices,
            routed.float() * topk_weights[token_indices, route_indices, None],
        )
    torch.testing.assert_close(output, output_ref.to(dtype), rtol=0.04, atol=0.06)
```

python/sglang/kernels/jit/csrc/gemm/marlin_moe/moe_wna16_marlin.cuh

实现 occupancy-aware launch 配置选择逻辑的辅助文件，新增 register/shared-memory 计算和允许并发 block 数估算，是性能提升的关键逻辑。

```
// moe_wna16_marlin.cuh（occupancy 选择逻辑）

// 共享内存有效性裕量和发射预留（避免共享内存不足导致的 invalid 配置）
constexpr int kSharedMemoryValidityMargin = 512;
constexpr int kSharedMemoryLaunchReserve = 1024;

template <typename scalar_t, bool kIsEP, bool kHasBias>
exec_config_t determine_exec_config(
    const host::ScalarType& q_type, int prob_m, int prob_n, int prob_k,
    int top_k, int thread_m_blocks, bool m_block_size_8,
    int num_bits, int group_size, bool is_k_full, bool has_zp, bool is_zp_float,
    int max_shared_mem, int sms) // 新增 sms 参数（当前未直接使用）
{
    exec_config_t exec_cfg = {1, {-1, -1, -1}};
    // ... 配置遍历 ...
    for (auto& th_config : thread_configs) {
        // ... 计算 cache_size, group_blocks 等 ...
        auto kernel = get_marlin_kernel<scalar_t, kIsEP, kHasBias>(...);
```

```
        if (kernel == MarlinDefault) continue;

        // 获取 kernel 属性：寄存器数和共享内存使用量
        cudaFuncAttributes attr;
        cudaFuncGetAttributes(&attr, kernel);
        int reg_size = max(attr.numRegs, 1) * th_config.num_threads * 4;

        // 估算允许的并发 block 数（受寄存器文件和共享内存容量限制）
        int allow_count = min(
            device_max_reg_size / reg_size,
            max_shared_mem / (cache_size + kSharedMemoryValidityMargin +
            kSharedMemoryLaunchReserve)
        );
        allow_count = max(min(allow_count, thread_m_blocks == 1 ? 4 : 2), 1);

        if (thread_m_blocks > 1) {
            // large-batch：取第一个有效配置（简化处理）
            exec_cfg = {1, th_config};
            break;
        } else {
            // small-batch：选择 allow_count 最大的配置
            if (allow_count > count) {
                count = allow_count;
                exec_cfg = {count, th_config};
            }
        }
    }
    return exec_cfg;
}
```

评论区精华

Review 中 BBuf 询问模型精度结果（"Any model acc results can be added?"），作者在 PR body 中补充了 GSM8K 200 例 5-shot 对比：基线 97.5% vs. 变更后 97.0%（-0.5 pp，1 例差异），确认无实质性退化。无其他争议，BBuf 最终批准。

- 模型精度验证 (correctness): 作者在 PR body 中补充 GSM8K 对比：基线 97.5% vs. 变更后 97.0%（-0.5 pp），确认无显著退化。

风险与影响

- 风险：
  1. 模型精度风险：GSM8K 准确率下降 0.5 个百分点（200 例中错 1 例），虽在合理范围内，但更全面的评测（如多个数据集）尚未覆盖。
  2. JIT 编译实例膨胀：kIsEP 和 kHasBias 组合增加编译缓存条目数，首次启动可能稍慢，但后续重用无影响。

3. 核心 kernel 修改：marlin_template.h 和 moe_wna16_marlin.cuh 是敏感路径，修改可能影响其他调用方（如 NVFP4 变体），但本 PR 保持接口兼容且引入的 if constexpr 在非 EP/ 无 bias 路径下与之前等价。
4. 共享内存 margin 调整：is_valid_config 中 margin 从 512 改为 kSharedMemoryValidityMargin（仍为 512），并增加 kSharedMemoryLaunchReserve 用于 occupancy 计算，可能改变某些边缘 case 的配置选择。 - 影响：影响范围限于使用 moe_wna16_marlin 推理入口的模型（主流量化 MoE 路径，如 Kimi-K2.7）。对非 Marlin MoE 路径无影响。性能提升在 H200 + TP8 上已验证 prefill 加速约 4%，TTFT 加速约 3.5%。团队可获得 occupancy 感知调度与编译时专门化的参考实现。 - 风险标记：模型精度轻微下降（0.5%），JIT 编译实例膨胀，核心 kernel 路径修改，共享内存 margin 调整影响配置选择

关联脉络

- PR #32248 Migrate CompressedTensorsW4A4Nvfp4MoE TRT-LLM path onto MoeRunner: 同为 Marlin MoE 量化推理路径的基础设施重构，本 PR 在其框架下进行性能优化。