PR #23754 完整报告

sgl-project/sglang

[Quantization] add humming quantization kernel

合并时间：2026-07-14 08:42

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23754

执行摘要

- 一句话：集成 Humming 量化内核，提升 MoE 推理性能
- 推荐动作：值得精读，尤其是 quantization/humming.py 的量化方法选择逻辑和 moe_runner/humming.py 的 GEMM 类型调度。设计上的权衡（fallback、lazy import、layer 引用传递）对其他后端集成具有参考意义。建议跟进者关注即将到来的测试补充和依赖可选化议题。

功能与动机

Humming 是 InclusionAI 开发的高性能量化内核，对标 Marlin 但支持更多位宽、更优的 Hopper 性能以及更快的 JIT 编译。官方 README 和 vLLM PR#34556 已证明其收益。本 PR 填补了 SGLang 对 Humming 的支持空白，尤其为 DeepSeek V4 的高效推理提供了 W4A8 方案。

实现拆解

实现分为以下几步：

1. 量化方法与配置（quantization/humming.py）：新增 HummingConfig 类，提供 Humming 特有的参数定义，并实现 lazy import 以避免运行时依赖。
2. 权重预处理（quantization/humming_utils.py）：prepare_humming_layer 和 prepare_humming_moe_layer 负责在加载后转换权重布局，插入 pad 和 transform。
3. MoE Runner（moe/moe_runner/humming.py）：实现 HummingRunnerCore，支持 grouped/indexed/masked 三种 GEMM 类型，并通过 humming_moe_runner_core_run 自定义算子分发。
4. 融合后处理核（moe/fused_moe_triton/moe_fused_mul_sum.py）：新增 Triton 核 moe_fused_mul_sum，用于带 EP 和 expert_map 的加权求和。
5. DeepSeek V4 支持（quantization/mxfp4_humming_moe.py）：Mxfp4HummingMoEMethod 桥接 FP8 基础方法和 Humming MoE Runner，在权重加载后将 scale 转为 E8M0 并调用 prepare_humming_moe_layer。
6. 环境变量与参数（environ.py, server_args.py, pyproject.toml）：添加 SGLANG_HUMMING_* 环境变量、--quantization humming 运行时参数，以及可选依赖 humming-kernels[cu13]。

7. JIT 内核迁移（jit_kernel/moe_permute_prepare.py）：将 moe_permute_prepare CUDA 核包装为 SGLang JIT 模块，避免编译进主库。
8. EP 适配（ep_moe/layer.py）：在 EP MoE 层中识别 Humming，自动开启 BF16 分发以保持精度。

关键文件：

- python/sglang/srt/layers/quantization/humming.py（模块 量化层；类别 source；类型 core-logic；符号 _lazy_import_humming, prepare_padded_shape, prepare_param, prepare_moe_param）：核心量化方法文件，定义了 HummingConfig 和参数准备，是 Humming 集成的入口
- python/sglang/srt/layers/moe/moe_runner/humming.py（模块 MoE 调度；类别 source；类型 core-logic；符号 get_standard_humming_moe_gemm_type, HummingRunnerInput, runner_backend, HummingRunnerOutput）：Humming MoE 调度核心，实现 runner 和 GEMM 类型分发
- python/sglang/srt/layers/moe/fused_moe_triton/moe_fused_mul_sum.py（模块 融合核；类别 source；类型 core-logic；符号 moe_fused_mul_sum_kernel, _heuristic_config, moe_fused_mul_sum）：新增融合乘加 Triton 核，用于 EP/expert_map 场景的加权求和
- python/sglang/srt/layers/quantization/mxfp4_humming_moe.py（模块 MXFP4 桥接；类别 source；类型 core-logic；符号 Mxfp4HummingMoEMethod, init, create_moe_runner, create_weights）：DeepSeek V4 FP4 专家桥接到 Humming MoE runner
- python/sglang/srt/layers/quantization/humming_utils.py（模块 量化工具；类别 source；类型 core-logic；符号 humming_is_layer_skipped, prepare_humming_layer, prepare_humming_moe_layer）：提供 Humming 层的权重准备和布局转换工具
- python/sglang/jit_kernel/moe_permute_prepare.py（模块 JIT 内核；类别 source；类型 dependency-wiring；符号 _jit_moe_permute_prepare_module, _moe_permute_prepare_out, moe_permute_prepare）：将 MoE permute prepare 内核包装为 JIT 模块，避免静态编译
- python/sglang/srt/layers/moe/ep_moe/kernels.py（模块 EP 内核；类别 source；类型 core-logic；符号 moe_permute, moe_unpermute）：修改 moe_permute/moe_unpermute 以支持 Humming 的专家映射
- python/sglang/srt/environ.py（模块 环境配置；类别 source；类型 configuration；符号 SGLANG_HUMMING_ONLINE_QUANT_CONFIG, SGLANG_HUMMING_INPUT_QUANT_CONFIG, SGLANG_HUMMING_USE_F16_ACCUM, SGLANG_HUMMING_MOE_GEMM_TYPE）：添加 Humming 相关的 4 个环境变量
- python/sglang/srt/server_args.py（模块 服务参数；类别 source；类型 configuration）：添加 --quantization humming 选项
- python/pyproject.toml（模块 依赖配置；类别 config；类型 configuration）：添加 humming-kernels[cu13] 作为运行时依赖

关键符号：_lazy_import_humming, prepare_padded_shape, prepare_param, prepare_moe_param, may_pad_loaded_weight, compressed_tensors_get_config, HummingConfig, get_standard_humming_moe_gemm_type, HummingRunnerInput, HummingRunnerOutput, HummingMoeQuantInfo, humming_moe_runner_core_run, HummingRunnerCore, moe_fused_mul_sum_kernel, _heuristic_config, moe_fused_mul_sum, Mxfp4HummingMoEMethod, humming_is_layer_skipped, prepare_humming_layer, prepare_humming_moe_layer, _jit_moe_permute_prepare_module, _moe_permute_prepare_out, moe_permute_prepare, moe_permute, moe_unpermute

关键源码片段

python/sglang/srt/layers/quantization/humming.py

核心量化方法文件，定义了 HummingConfig 和参数准备，是 Humming 集成的入口

```
# humming.py - Humming 量化核心模块

# 模块级变量，由 _lazy_import_humming 填充
DataType = None
HummingMethod = None
BaseInputSchema = None
BaseWeightSchema = None
HummingInputSchema = None
HummingWeightSchema = None
quantize_weight = None

def _lazy_import_humming():
    """延迟导入 Humming kernel 库，仅在首次需要时加载。
    若未安装 humming-kernels，则抛出带有安装提示的 ImportError。
    """
    global DataType, HummingMethod, BaseInputSchema, BaseWeightSchema
    global HummingInputSchema, HummingWeightSchema, quantize_weight

    if HummingMethod is not None: # 已导入，直接跳过
        return

    try:
        from humming.dtypes import DataType as _DataType
        from humming.layer import HummingMethod as _HummingMethod
        from humming.schema import BaseInputSchema as _BaseInputSchema
        from humming.schema import BaseWeightSchema as _BaseWeightSchema
        from humming.schema import HummingInputSchema as _HummingInputSchema
        from humming.schema import HummingWeightSchema as _HummingWeightSchema
        from humming.utils.weight import quantize_weight as _quantize_weight
    except ImportError as err:
        if isinstance(err, ModuleNotFoundError) and err.name == "humming":
            message = (
                "Humming quantization requires `humming-kernels`. "
```

```
                "Please install it to use `--quantization humming`."
            )
        else:
            message = f"Failed to import Humming quantization dependencies from `humming-
            kernels`: {err}"
        raise ImportError(message) from err

    DataType = _DataType
    HummingMethod = _HummingMethod
    BaseInputSchema = _BaseInputSchema
    BaseWeightSchema = _BaseWeightSchema
    HummingInputSchema = _HummingInputSchema
    HummingWeightSchema = _HummingWeightSchema
    quantize_weight = _quantize_weight

# --- 参数准备函数 ---

def prepare_param(tensor, name, extra_attrs):
    """将 Humming 格式的张量包装为 SGLang 参数类型。
    根据 extra_attrs 中的 scale_type 等属性选择合适的 Parameter 子类。
    """
    extra_attrs = extra_attrs.copy()
    scale_type = extra_attrs.pop("scale_type", None)
    param_cls_name_map = {
        "block": BlockQuantScaleParameter,
        "tensor": PerTensorScaleParameter,
        "group": GroupQuantScaleParameter,
        "channel": ChannelQuantScaleParameter,
        "input_scale": PerTensorScaleParameter,
    }

    if "packed_dim" in extra_attrs:
        param_cls = PackedvLLMParameter
    elif scale_type in param_cls_name_map:
        param_cls = param_cls_name_map[scale_type]
    elif "output_dim" in extra_attrs and "input_dim" in extra_attrs:
        param_cls = ModelWeightParameter
    elif "input_dim" in extra_attrs:
        param_cls = RowvLLMParameter
    elif "output_dim" in extra_attrs:
        param_cls = ChannelQuantScaleParameter
    else:
        param_cls = BasevLLMParameter

    kwargs_keys = ["input_dim", "output_dim", "packed_dim", "packed_factor", "weight_loader"]
    cls_kwargs = {k: extra_attrs.pop(k) for k in extra_attrs.copy() if k in kwargs_keys}

    param = param_cls(data=tensor, **cls_kwargs)
    set_weight_attrs(param, extra_attrs)
```

```
    param.param_name = name
    param.ignore_warning = True
    if scale_type in ["tensor", "input_scale"]:
        param.needs_scalar_to_array = True
    return param
```

python/sglang/srt/layers/moe/moe_runner/humming.py

Humming MoE 调度核心，实现 runner 和 GEMM 类型分发

```
# humming_moe_runner.py - Humming MoE 调度核心

@register_custom_op()
def humming_moe_runner_core_run(
    moe_runner_id: int,
    gemm_type: str,
    hidden_states: torch.Tensor,
    topk_weights: torch.Tensor,
    topk_ids: torch.Tensor,
    expert_num_tokens: torch.Tensor | None = None,
    expected_m: int | None = None,
    apply_routed_scaling_factor: bool = True,
) -> torch.Tensor:
    """通过自定义算子调用的 MoE 核心运行函数。
    根据 gemm_type 选择不同的 GEMM 实现路径。
    """
    runner = HummingRunnerCore.runner_cores[moe_runner_id] # 从弱引用字典获取 runner
    if gemm_type == "indexed":
        return runner._run_indexed_gemm(
            hidden_states=hidden_states,
            topk_ids=topk_ids,
            topk_weights=topk_weights,
            apply_routed_scaling_factor=apply_routed_scaling_factor,
        )
    elif gemm_type == "grouped_contiguous":
        return runner._run_grouped_contiguous_gemm(
            hidden_states=hidden_states,
            topk_ids=topk_ids,
            topk_weights=topk_weights,
            apply_routed_scaling_factor=apply_routed_scaling_factor,
        )
    elif gemm_type == "grouped_masked":
        assert expected_m is not None and expert_num_tokens is not None
        return runner._run_grouped_masked_gemm(
            hidden_states=hidden_states,
            topk_ids=topk_ids,
            topk_weights=topk_weights,
            expected_m=expected_m,
            expert_num_tokens=expert_num_tokens,
        )
```

```
    else:
        raise ValueError(f"Unknown gemm type: {gemm_type}")
```

评论区精华

Review 中重点讨论了以下问题：

- 屏障必要性：BBuf 质疑 moe_align_block_size_kernel 中新增的 __syncthreads()。作者确认是早期调试遗留，已移除。
- 依赖策略：BBuf 担心 humming-kernels 作为硬依赖会导致非 CUDA 环境安装失败。作者解释 Humming 包很轻量且 CUDA-only，最终 PR 保留了硬依赖但后续可转为 optional。
- Fallback 设计：BBuf 对 weight loading 时通过修改 self.__class__ 回退到 UnquantizedLinearMethod 表示担忧。作者说明这是为了防御某些未标记为忽略的小线性层，且实际较少触发。
- Layer 引用传递：OrangeRedeng 指出应将 layer 引用放入 HummingMoeQuantInfo 而非 MoeRunnerConfig。作者同意并实施了更改。
- 环境变量文档：OrangeRedeng 要求为新增的 4 个 Humming 环境变量添加文档。作者后续补上了文档。
  - moe_align_kernel 屏障必要性 (correctness): 作者确认是早期调试遗留，已移除该 __syncthreads()。
  - Humming 依赖策略 (design): 作者解释 Humming 包很轻量（~200KB）且依赖少，BBuf 最终同意保留硬依赖，但未来可能考虑 optional。
  - Weight loading fallback 设计 (design): 作者说明这是为了防御某些未在 modules_to_not_convert 中声明的小线性层（如 MoE gate），实际较少触发，但承认不是最佳做法。
  - Layer 引用传递 (design): 作者同意并将 layer 引用移入 HummingMoeQuantInfo，计划未来进一步抽象。
  - 环境变量文档 (documentation): 作者后续补上了文档（在 docs/references/environment_variables.md 中）。

风险与影响

- 风险：
  - 依赖风险：引入 humming-kernels[cu13] 作为核心依赖，非 CUDA 13 环境可能缺少二进制，但作者报告内核支持更广。未来应考虑做 optional。
  - 缺少测试覆盖：PR 没有添加对应的单元测试或集成测试（基准测试位于 PR 描述但非代码）。风险：片权重加载或 MoE runner 选择错误可能不被及时发现。
  - 配置复杂性：新增 4 个环境变量和多个 GEMM 类型选项，用户可能需要额外调优才能获得最佳性能。
  - GPU 兼容性：虽然 Humming 支持 Hopper 和 Ada，但性能调优主要面向 SM90+；在旧 GPU 上可能退化为非最优策略。

    - MoE runner 选择冲突：humming.py 中的 get_quant_method() 可能与其他量化方法重叠，需要确保 --moe-runner-backend 和 --quantization humming 的一致性。
- 影响：
    - 用户：新用户可通过 --quantization humming 启用，在合理 GPU 上获得显著性能提升（吞吐量 +10~30%, TTFT 降低）；但需安装 humming-kernels 依赖。
    - 系统：MoE runner 增加新后端，量化框架扩展，但不影响现有 Marlin/AWQ/GPTQ 路径。
    - 维护者：需要维护额外的量化方法，未来重构（如 #15194 式的拆包）可参考 AWQ 的前例。
    - 风险标记：外部依赖，缺少测试覆盖，配置复杂，核心路径变更

关联脉络

- PR #15194 [Quantization] Refactor quantization to use scheme-based approach: 评论中引用的量化重构方向，建议将 humming.py 拆分为 package 以保持架构一致性。
- PR #21126 [Quantization] Split AWQ into scheme-based quantization and backend: 作为 humming 拆分的参考 PR，展示了 scheme-based 的组织方式。