PR #24117 完整报告

sgl-project/sglang

[codex] Optimize Z-Image packed QKV

合并时间：2026-05-07 07:51

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24117

执行摘要

- 一句话：Z-Image 打包 QKV 投影优化，去噪延迟降低 35%
- 推荐动作：建议技术负责人和扩散模型开发者精读此 PR，特别是 linear.py 中 _weight_loader_v2_block_quant_scale 的实现，这是一个为融合线性层处理块量化权重的良好模式。未来类似模型（如 Flux/MMDiT）可借鉴此方案。

功能与动机

Z-Image 模型之前仅在加载 Nunchaku 量化检查点时使用融合的 QKV 投影（packed QKV），加载标准 BF16 检查点时仍使用分离的 Q/K/V 投影。此 PR 旨在将 packed QKV 的优势（减少内核启动次数、提升内存带宽利用率）扩展到标准检查点加载场景，从而显著降低推理延迟。

实现拆解

1. 配置层映射（data-contract）：在 python/sglang/multimodal_gen/configs/models/dits/zimage.py 的 ZImageArchConfig.param_names_mapping 中添加了 to_q/to_k/to_v.weight、weight_scale_inv 和 LoRA 张量到 to_qkv 的映射规则，使权重加载器在加载检查点时自动将三个独立的权重合并到单个 to_qkv 参数中。
2. 模型始终使用融合QKV：在python/sglang/multimodal_gen/runtime/models/dits/zimage.py 的 ZImageAttention.__init__ 中将 self.use_fused_qkv 从 isinstance(quant_config, NunchakuConfig) 改为始终为 True，因此无论是否量化，都创建 MergedColumnParallelLinear 而非三个独立线性层。
3. 新增 BlockQuantScaleParameter 加载方法：在 python/sglang/multimodal_gen/runtime/layers/linear.py 的 MergedColumnParallelLinear 中，weight_loader_v2 方法首先检查参数是否为 BlockQuantScaleParameter，若是则调用新方法 _weight_loader_v2_block_quant_scale。该方法处理分片加载、块缩放偏移计算，并使用 divide 确保对齐。同时移除了之前占位的 raise NotImplementedError。
4. 配套测试：未添加新的测试文件；PR 作者通过手动运行基准脚本和原生后端门控日志（检查 fallback 字符串）验证了功能正确性。

关键文件：

- python/sglang/multimodal_gen/runtime/layers/linear.py（模块 核心层；类别 source；类型 core-logic；符号 _weight_loader_v2_block_quant_scale）：核心变更：新增 _weight_loader_v2_block_quant_scale 方法，替换原有的 NotImplementedError，实现

BlockQuantScaleParameter 的块级权重加载。同时修改 weight_loader_v2 的控制流，优先处理 BlockQuantScaleParameter。

- python/sglang/multimodal_gen/configs/models/dits/zimage.py（模块 配置层；类别 source；类型 data-contract）：新增 param_names_mapping 规则，将检查点中的 to_q/to_k/to_v 权重（及其 scale_inv 和 LoRA 变体）映射到 to_qkv 参数。这是启用 packed QKV 的数据契约基础。
- python/sglang/multimodal_gen/runtime/models/dits/zimage.py（模块 模型定义；类别 source；类型 data-contract）：一行关键改动：将 self.use_fused_qkv 从 isinstance(quant_config, NunchakuConfig) 改为 True，使得始终使用 MergedColumnParallelLinear 结构的 to_qkv。

关键符号：_weight_loader_v2_block_quant_scale, weight_loader_v2

关键源码片段

python/sglang/multimodal_gen/runtime/layers/linear.py

核心变更：新增 _weight_loader_v2_block_quant_scale 方法，替换原有的 NotImplementedError，实现 BlockQuantScaleParameter 的块级权重加载。同时修改 weight_loader_v2 的控制流，优先处理 BlockQuantScaleParameter。

```
# python/sglang/multimodal_gen/runtime/layers/linear.py (partial)
# 在 MergedColumnParallelLinear.weight_loader_v2 中新增的 BlockQuantScaleParameter 分支
def weight_loader_v2(
    self,
    param: BasevLLMParameter,
    loaded_weight: torch.Tensor,
    loaded_shard_id: int | None = None,
) -> None:
    # 新增：如果参数是 BlockQuantScaleParameter，委托给专用方法
    if isinstance(param, BlockQuantScaleParameter):
        self._weight_loader_v2_block_quant_scale(
            param, loaded_weight, loaded_shard_id
        )
        return

    # 原有逻辑保持不变 ...

# 新增的专用加载方法
def _weight_loader_v2_block_quant_scale(
    self,
    param: BlockQuantScaleParameter,
    loaded_weight: torch.Tensor,
    loaded_shard_id: int | None = None,
) -> None:
    assert self.quant_method is not None
    weight_block_size = getattr(
        self.quant_method.quant_config, "weight_block_size", None
```

```
)
if weight_block_size is None:
    raise ValueError(
        "MergedColumnParallelLinear block-scale loading requires "
        "quant_config.weight_block_size."
    )
block_n, _ = weight_block_size # 块大小，例如 128
output_dim = param.output_dim

if loaded_shard_id is None:
    # 无分片 ID：要么整个权重形状匹配直接拷贝，要么按 output_sizes 遍历分片
    if param.data.shape == loaded_weight.shape:
        param.data.copy_(loaded_weight)
        return
    block_offset = 0
    for shard_id, output_size in enumerate(self.output_sizes):
        block_size = divide(output_size, block_n) # 块对齐后的分片大小
        loaded_weight_shard = loaded_weight.narrow(
            output_dim, block_offset, block_size
        )
        self._weight_loader_v2_block_quant_scale(
            param, loaded_weight_shard, shard_id
        )
        block_offset += block_size
    return

# 有分片 ID：计算当前分片的块偏移和大小
assert loaded_shard_id < len(self.output_sizes)
shard_offset = divide(
    sum(self.output_sizes[:loaded_shard_id]), self.tp_size
)
shard_size = divide(
    self.output_sizes[loaded_shard_id], self.tp_size
)
block_shard_offset = divide(shard_offset, block_n)
block_shard_size = divide(shard_size, block_n)

# 从 param.data 中切出目标区域
param_data = param.data.narrow(
    output_dim, block_shard_offset, block_shard_size
)
# 当前 rank 需要加载的部分
start_idx = self.tp_rank * block_shard_size
loaded_weight = loaded_weight.narrow(
    output_dim, start_idx, block_shard_size
)
assert param_data.shape == loaded_weight.shape
param_data.copy_(loaded_weight)
```

python/sglang/multimodal_gen/configs/models/dits/zimage.py

新增 param_names_mapping 规则，将检查点中的 to_q/to_k/to_v 权重（及其 scale_inv 和 LoRA 变体）映射到 to_qkv 参数。这是启用 packed QKV 的数据契约基础。

```
# python/sglang/multimodal_gen/configs/models/dits/zimage.py
# ZImageArchConfig 中新增的 param_names_mapping 条目
param_names_mapping: dict = field(
    default_factory=lambda: {
        # 将三个分离的权重映射到融合的 to_qkv
        r"(.*)\.attention\.to_q\.weight$": (r"\1.attention.to_qkv.weight", 0, 3),
        r"(.*)\.attention\.to_k\.weight$": (r"\1.attention.to_qkv.weight", 1, 3),
        r"(.*)\.attention\.to_v\.weight$": (r"\1.attention.to_qkv.weight", 2, 3),
        # 也处理量化缩放参数（block scale）
        r"(.*)\.attention\.to_q\.weight_scale_inv$": (r"\1.attention.to_qkv.weight_scale_inv", 0, 3),
        r"(.*)\.attention\.to_k\.weight_scale_inv$": (r"\1.attention.to_qkv.weight_scale_inv", 1, 3),
        r"(.*)\.attention\.to_v\.weight_scale_inv$": (r"\1.attention.to_qkv.weight_scale_inv", 2, 3),
        # 处理 LoRA 适配器
        r"(.*)\.attention\.to_q\.(lora_A|lora_B)$": (r"\1.attention.to_qkv.\2", 0, 3),
        r"(.*)\.attention\.to_k\.(lora_A|lora_B)$": (r"\1.attention.to_qkv.\2", 1, 3),
        r"(.*)\.attention\.to_v\.(lora_A|lora_B)$": (r"\1.attention.to_qkv.\2", 2, 3),
        # 原有前馈映射保持不变 ...
        r"(.*)\.feed_forward\.w1\.weight$": (r"\1.feed_forward.w13.weight", 0, 2),
        r"(.*)\.feed_forward\.w3\.weight$": (r"\1.feed_forward.w13.weight", 1, 2),
    }
)
```

评论区精华

Review 中获得 mickqian 的批准（APPROVED），无额外评论。PR 在 commit 历史中有多次合并 main 分支的操作，但最终提交无争议。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 量化路径兼容性风险（linear.py）：新添加的 _weight_loader_v2_block_quant_scale 方法替换了之前的 raise NotImplementedError。如果 Nunchaku 量化路径使用 BlockQuantScaleParameter，新逻辑必须正确处理 block scale 的分片。当前实现通过 quant_config.weight_block_size 获取块大小，但若 quant_config 为 None 或缺少 weight_block_size 会引发 ValueError。需确认所有使用 BlockQuantScaleParameter 的场景均满足此前提。
  2. 无测试覆盖：本次变更没有新增单元测试或集成测试，回归风险依赖人为基准测试。若未来有代码重构，该逻辑可能被意外破坏。
  3. 性能假阳性：基准测试结果基于特定硬件（H200）和 prompt，其他配置下性能提升可能略有不同，但方向应一致。 - 影响：影响范围：仅影响 Z-Image 扩散模型的推理路径。用户升级后，加载标准 BF16 检查点会自动启用 fused QKV，获得 30%+ 的延迟降低。

量化路径的行为不变（之前已使用 fused QKV）。影响程度：正面性能改进，无 API 或功能破坏。 - 风险标记：核心路径变更，缺少测试覆盖

关联脉络

- PR #24332 [Codex] Diffusion handle non-contiguous CFG communication: 同一作者（BBuf）的扩散模型相关 PR，涉及 Z-Image 的分布式通信兼容性修复。
- PR #23335 Fix diffusion fallback guards and validation: 之前的扩散模型修复，也修改了 Z-Image 相关的文件路径（jit_kernel）。本 PR 没有直接依赖，但属于同一功能线。