PR #26733 完整报告

sgl-project/sglang

Nemotron perf changes

合并时间：2026-06-06 13:31

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26733

执行摘要

- 一句话：Nemotron 模型推理性能显著提升
- 推荐动作：值得精读，尤其是 scaling factor 融合与 BF16 路由 GEMM 的设计模式，以及 JIT 激活算子如何统一派发。对于涉及 MoE 量化的团队，可借鉴其条件路由缩放的处理方式。

功能与动机

PR 旨在解决 Nemotron 模型推理中明显的性能瓶颈，包括不合适的 attention 后端选择、路由器 FP32 计算开销、scaling factor 额外乘法、ReLU2 算子未融合、以及多余的内存拷贝等问题。通过针对性优化，显著提升部署效率。

实现拆解

1. 新增 ReLU2 JIT 内核(python/sglang/jit_kernel/activation.py, python/sglang/srt/layers/activation.py): 在 JIT 激活模块中注册 run_unary_activation 和 relu2 接口，ReLU2 类从 nn.Module 改为 MultiPlatformOp，CUDA 路径派发到 JIT kernel。
2. 路由 GEMM 精度调整(python/sglang/srt/models/nemotron_h.py): 将路由器计算由 gate(hidden_states.float32) 改为 torch.mm(hidden_states, gate.weight.t(), out_dtype=torch.float32)，使得矩阵乘法在 BF16 上进行但输出 FP32，减少 FP32 计算量。
3. Scaling Factor 融合(python/sglang/srt/models/nemotron_h.py, python/sglang/srt/layers/quantization/unquant.py): 将 routed_scaling_factor 传递给 MoE 专家模块，并让 TopK 层通过 apply_routed_scaling_factor_on_output 标志决定是否在路由概率中融合缩放；同时在未融合时由量化层在输出端应用，避免双重缩放。移除了 forward() 中冗余的 final_hidden_states *= self.routed_scaling_factor。
4. Mamba 性能优化(python/sglang/srt/layers/attention/mamba/mamba.py, layernorm_gated.py, causal_conv1d_triton.py): 将 Mamba2 的 layer norm 融合扩大到支持 num_groups > 1，减少 kernel launch 次数；消除 out_proj 写入时的多余内存拷贝。
5. MoE 后端自动选择(python/sglang/srt/arg_groups/nemotron_h_hook.py): 在 unquantized (BF16) 或 ModelOpt 量化时，优化 MoE runner 后端的选择策略，默认使用 flashinfer_trtllm 或 marlin 等高效后端。

6. 配套测试与基准(python/sglang/jit_kernel/tests/test_activation.py, benchmark/bench_activation.py): 新增 ReLU2 的精度测试和 performance benchmark, 覆盖多种 shape 和 dtype。

关键文件:

- python/sglang/jit_kernel/activation.py（模块 JIT 内核；类别 source；类型 core-logic；符号 _run_unary_activation_inplace, run_unary_activation, relu2）：核心变更：新增 unary activation 框架，注册 run_unary_activation 和 relu2 接口，为 JIT kernel 添加 run_unary_activation wrapper。
- python/sglang/srt/models/nemotron_h.py（模块 模型实现；类别 source；类型 data-contract）：关键模型端变更：路由器 GEMM 切换为 BF16，scaling factor 融合进 TopK 和专家层，消除后处理乘法。
- python/sglang/srt/layers/activation.py（模块 算子层；类别 source；类型 core-logic；符号 ReLU2, forward, forward_native, forward_cuda）：ReLU2 算子重构：继承 MultiPlatformOp 以利用多平台派发，CUDA 路径转发到 JIT kernel。
- python/sglang/srt/arg_groups/nemotron_h_hook.py（模块 配置钩子；类别 source；类型 core-logic）：MoE 后端选择策略调整：在未量化或 ModelOpt 量化时重选后端，影响性能关键路径。
- python/sglang/jit_kernel/tests/test_activation.py（模块 激活函数；类别 test；类型 test-coverage；符号 test_relu2_correctness, test_relu2_out_param, test_relu2_negative_inputs_zeroed）：测试覆盖：新增 test_relu2_correctness、test_relu2_out_param、test_relu2_negative_inputs_zeroed 三个测试函数，确保 ReLU2 kernel 正确性。
- python/sglang/jit_kernel/benchmark/bench_activation.py（模块 JIT 内核；类别 source；类型 core-logic；符号 relu2_torch, benchmark_unary）：基准：新增 benchmark_unary 函数，对比 JIT kernel 与 torch compile 实现的 ReLU2 性能。

关键符号：run_unary_activation, relu2, ReLU2.forward_cuda, NemotronHMoE._forward_core_normal, NemotronHMoE.forward, apply_nemotron_h_defaults

关键源码片段

python/sglang/jit_kernel/activation.py

核心变更：新增 unary activation 框架，注册 run_unary_activation 和 relu2 接口，为 JIT kernel 添加 run_unary_activation wrapper。

```
# python/sglang/jit_kernel/activation.py (head)

SUPPORTED_UNARY_ACTIVATIONS = {"relu2"} # 新增单输入激活集合

@register_custom_op(mutates_args=["out"])
def _run_unary_activation_inplace(
    op_name: str, input: torch.Tensor, out: torch.Tensor
) -> None:
```

```
    # 单输入激活：input 和 out 形状相同，无 gate/up 拆分
    last = input.shape[-1]
    module = _jit_activation_module(input.dtype)
    module.run_unary_activation(input.view(-1, last), out.view(-1, last), op_name)

def run_unary_activation(
    op_name: str,
    input: torch.Tensor,
    out: Optional[torch.Tensor] = None,
) -> torch.Tensor:
    """Apply a standalone element-wise activation: out = act(input)"""
    assert op_name in SUPPORTED_UNARY_ACTIVATIONS, f"Unsupported: {op_name}"
    if out is None:
        out = torch.empty_like(input)
    _run_unary_activation_inplace(op_name, input, out)
    return out

def relu2(input, out=None):
    """Squared ReLU: out = max(0, input) ** 2"""
    return run_unary_activation("relu2", input, out)
```

python/sglang/srt/models/nemotron_h.py

关键模型端变更：路由器 GEMM 切换为 BF16，scaling factor 融合进 TopK 和专家层，消除后处理乘法。

```
# python/sglang/srt/models/nemotron_h.py (head) def_forward_core_normal(self, hidden_states):    # 路由 GEMM：使用 BF16 乘法 +FP32 累加（out_dtype），减少 FP32 计算量    router_logits = torch.mm(        hidden_states, self.gate.weight.t(), out_dtype=torch.float32    )    if self.shared_experts is not None:        shared_output = self.shared_experts(hidden_states)    else:        shared_output = None topk_output = self.topk(hidden_states, router_logits)    if self.use_latent_moe:        hidden_states, _ = self.fc1_latent_proj(hidden_states)    final_hidden_states = self.experts(hidden_states, topk_output)    return final_hidden_states, shared_output defforward(self, hidden_states):    # scaling factor 已由 TopK 或 experts 内部融合，此处不再手动缩放    final_hidden_states, shared_output = self._forward_core(hidden_states)    # 注意：原来有 final_hidden_states *= self.routed_scaling_factor，现已移除    num_tokens, hidden_dim = hidden_states.shape    if self.shared_experts is not None:        # shared experts 缩放        output = torch.empty(num_tokens, hidden_dim, ...)        output[:num_tokens] = final_hidden_states + shared_output * (1.0 / self.routed_scaling_factor)    else:        output = final_hidden_states    return output （注意：此处为整理后的示意代码，实际实现更复杂）
```

python/sglang/srt/layers/activation.py

ReLU2 算子重构：继承 MultiPlatformOp 以利用多平台派发，CUDA 路径转发到 JIT kernel。

```
# python/sglang/srt/layers/activation.py (head)
```

```
from sglang.jit_kernel.activation import relu2 as _jit_relu2

class ReLU2(MultiPlatformOp):
    """
    Applies the squared Rectified Linear Unit function.
    y = max(0, x)^2
    """
    def forward_native(self, x: torch.Tensor) -> torch.Tensor:
        x = F.relu(x)
        return x * x

    def forward_cuda(self, x: torch.Tensor) -> torch.Tensor:
        # CUDA 路径使用 JIT kernel 实现
        return _jit_relu2(x)

    # forward_hip, forward_cpu 等可继承 MultiPlatformOp 默认行为
```

评论区精华

- Scaling factor 融合意图(Fridge003@nemotron_h.py#285): 作者 b8zhong 确认融合是故意的，因为乘法可以在融合计算中在 FP32 中进行，现有测试通过。
- Quantization 为 None 的含义(Fridge003@nemotron_h_hook.py#34): b8zhong 解释 None 表示未量化（BF16），但承认命名不够清晰。
- kUsePDL 模板控制(Fridge003@activation.cuh#206): b8zhong 说明 kUsePDL 在编译时通过 make_cpp_args 确定，而非运行时参数。
- Mamba out_proj 修改原因(Fridge003@mamba.py#725): 作者未在 thread 中直接回复，可能是为了消除不必要的内存拷贝。
- UnQuant 路径的 scaling factor 处理(Fridge003@unquant.py#515): b8zhong 解释当 backend 未融合 SF 时，量化层在输出端应用，避免重复。
- FlashInfer SSU 扩展讨论(nvpohanh@issue): 建议向 FlashInfer 团队提议支持 topk>1 的 SSU 场景。
- Scaling factor 融合意图 (design): 设计确认：融合是安全的，且精度不变。
- Quantization 为 None 的含义 (question): 澄清：None 代表 BF16 无量化。
- kUsePDL 模板控制 (question): 编译时模板参数，非运行时。
- Mamba out_proj 写入修改 (design): 性能优化：避免来自 PyTorch slicing 的内存拷贝。
- UnQuant 路径 scaling factor 处理 (design): 通用设计：保持与原来行为一致。

风险与影响

- 风险：
  - 路由 GEMM 精度：BF16 乘法可能导致精度损失，但作者声称准确性测试通过（GPQA 分数相近），仍需关注长序列 / 边缘案例。
  - JIT kernel 正确性：新增的 ReLU2 kernel 有单元测试覆盖，但缺乏对比 torch 编译路径的一致性。

    - Scaling factor 融合：可能影响非 Nemotron 模型（如 DeepSeek 等），需确保 should_fuse_routed_scaling_factor_in_topk 在相关 backend 中正确实现。
    - Mamba 路径修改：layernorm_gated.py 放宽了 n_groups 约束，可能影响其他使用 shared experts 或组路由的模型。
    - 平台限制：JIT kernel 仅支持 CUDA，其他加速器（AMD, Intel）需回退到 native 实现。
- 影响：
    - 用户：Nemotron 模型推理吞吐提升约 44%，延迟显著降低；BF16 路径默认启用，无需用户额外配置。
    - 系统：新增 run_unary_activation 统一接口，为后续其他单输入激活函数（如 ReLU、Sigmoid）的 JIT 化建立模式。
    - 团队：多个文件的耦合变更需要维护者确保跨模型一致性；新增的批量 benchmark 和精度测试有助于质量保障。
    - 风险标记：路由 BF16 精度，JIT kernel 正确性，融合 scaling factor 边效应，Mamba 通用路径影响，平台依赖（CUDA only）

关联脉络

- PR #27284 [CI] Fix Nemotron nightly mixed precision checkpoints test: 同一模型（Nemotron）的 CI 修复，维持 nightly 测试稳定性。