OP 的开发文档

Op接入资料&规范：开发备注

_copy_from

Function 接口

子接口 1fused_add_rms_norm

contiguous，也就是将 src tensor 中的内容重新排布后拷贝到 dst，保证 dst 张量的元素在内存中的布局是线性、顺序的，并且每个元素在内存中的地址都是连续的，也就是 *contiguous* 的

> 一个 PyTorch 张量 T 是 *contiguous* 的，当且仅当其步幅（stride）满足以下条件：
>
> 对于每个维度 i 来说，步幅 $\text{stride}(i)$ 满足：
>
> $$\text{stride}(i) = \prod_{j=i+1}^{n-1} \text{size}(j)$$
>
> 其中：
>
> - n 是张量的维度数。
> - $\text{size}(j)$ 是张量在第 j 个维度的大小（shape）。
> - $\text{stride}(i)$ 表示张量在维度 i 上相邻元素之间的内存步长。
>
> 这个表达式意味着元素在内存中按行优先顺序排列。如果不满足此条件，张量就是非连续的。

接口调用方需要保证 src 与 dst tensor 的 shape 一致

代码块

```
/**
dims: dst tensor dims
sizes: output tensor sizes
strides: strides of nargs tensors, 0 means it's a broadcasted dim
element_size: element_size of tensor
 **/
void fn(dev, const void* dst, const void* self, int dims, const int64_t*
sizes, const int64_t* const* strides, const int64_t element_size)

// 可选的形式(updated)
```

```
void fnv2(dev, const void* dst, const void* self, const int64_t self_ndims,
const int64_t* self_sizes, const int64_t* dst_strides, const int64_t*
self_strides, const int64_t element_size)
```

一些例子

代码块

```
// case 1
// python
b = torch.randn(4, 3).long().cuda()
b = b.t()
c = b.contiguous()
// kernel
copy_kernel_cuda: dst_device: cuda:0, src_device: cuda:0
copy_kernel_cuda: dst_sizes: [3, 4], src_sizes: [3, 4]
copy_kernel_cuda: dst_strides: [4, 1], src_strides: [1, 3]
copy_kernel_cuda: iter_shape: [4, 3]
copy_kernel_cuda: iter_dst_strides: [8, 32], iter_src_strides: [24, 8]

// case 2
// python
    a = torch.randn(3).long().cuda()
    b = torch.randn(4, 3).long().cuda()
    a = a.expand_as(b).contiguous()
// kernel
copy_kernel_cuda: dst_device: cuda:0, src_device: cuda:0
copy_kernel_cuda: dst_sizes: [4, 3], src_sizes: [4, 3]
copy_kernel_cuda: dst_strides: [3, 1], src_strides: [0, 1]
copy_kernel_cuda: iter_shape: [3, 4]
copy_kernel_cuda: iter_dst_strides: [8, 24], iter_src_strides: [8, 0]

// case3
// python
    a = torch.randn(4, 3, 2).long().cuda()
    b = torch.randn(1, 3, 1).long().cuda()
    a.copy_(b)
 // kernel
copy_kernel_cuda: dst_device: cuda:0, src_device: cuda:0
copy_kernel_cuda: dst_sizes: [4, 3, 2], src_sizes: [1, 3, 1]
copy_kernel_cuda: dst_strides: [6, 2, 1], src_strides: [3, 1, 1]
copy_kernel_cuda: iter_shape: [2, 3, 4]
copy_kernel_cuda: iter_dst_strides: [8, 16, 48], iter_src_strides: [0, 8, 0]

// case4
// python
```

```
39      a = torch.randn(4, 3, 2).long().cuda()
40      b = torch.randn(3, 1).long().cuda()
41      a.copy_(b)
42    // kernel
43  copy_kernel_cuda: dst_device: cuda:0, src_device: cuda:0
44  copy_kernel_cuda: dst_sizes: [4, 3, 2], src_sizes: [3, 1]
45  copy_kernel_cuda: dst_strides: [6, 2, 1], src_strides: [1, 1]
46  copy_kernel_cuda: iter_shape: [2, 3, 4]
47  copy_kernel_cuda: iter_dst_strides: [8, 16, 48], iter_src_strides: [0, 8, 0]
```

torch 中的调用路径：

aten/src/ATen/native/Copy.cpp copy_impl -> copy_stub -> aten/src/ATen/native/cuda/Copy.cu copy_kernel_cuda -> aten/src/ATen/native/cuda/Copy.cu copy_device_to_device -> aten/src/ATen/native/cuda/Copy.cu direct_copy_kernel_cuda -> torch/include/ATen/native/cuda/Loops.cuh gpu_kernel -> torch/include/ATen/native/cuda/CUDALoops.cuh gpu_kernel_impl -> torch/include/ATen/native/cuda/CUDALoops.cuh gpu_kernel_impl_nocast

ISSUE

- [ ] dtype 可能不一致，需要 kernel 做转换，转换的参数如何定义

_copy_from_and_resize，_copy_from_and_resize.out

参考信息

采用CPUFallback机制实现，通过 copy_impl 中的逻辑——先保证尺寸匹配、连续性，再调用通用的 copy_stub 内核，可以参考copy_from的实现 OP 的开发文档

ISSUE

_softmax

_softmax.out

参考信息

OP List 链接：https://siorigin.feishu.cn/record/UC8trohn4eZvxBc4JAJcGsGAn6b

Pytorch 官方文档参考：https://pytorch.org/docs/stable/generated/torch.nn.Softmax.html

Function 接口

```
代码块
1
```

```
2  void fn(dev, out*, input*,  bool half_to_float,  int64_t outer_size, dim_size,
   inner_size)
3  //(updated)
4  void fn(dev,const void* out,const void* self,  bool half_to_float, int64_t
   outer_size, dim_size, inner_size)
5  // 其中dim_size是dim参数所在维度的size，outer size是所在维度小于dim参数的所有维度大
   小:
6  // for (int64_t i = 0; i < dim; ++i)
7  //     outer_size *= input.size(i);
8  // 例如对于shape为[A,B,C,D,E]的Tensor在C维度做Softmax，会view成[A]
```

- 其中 dim_size 是 dim 参数所在维度的 size，outer size 是所在维度小于 dim 参数的所有维度大小：
  例如对于 shape 为[A，B，C，D，E]的 Tensor 在 C 维度做 Softmax，会 view 成[A*B，C，D*E]，对应的 stride 为[C*D*E，D*E，1]，outer_size=A*B，inner_size=D*E，dim_size=C
- half_to_float 代表是否要将 HALF 的 input 转换成 FLOAT（可以不支持 BFLOAT16）
- Accumulate 的 Dtype 参考：OP Summary

ISSUE

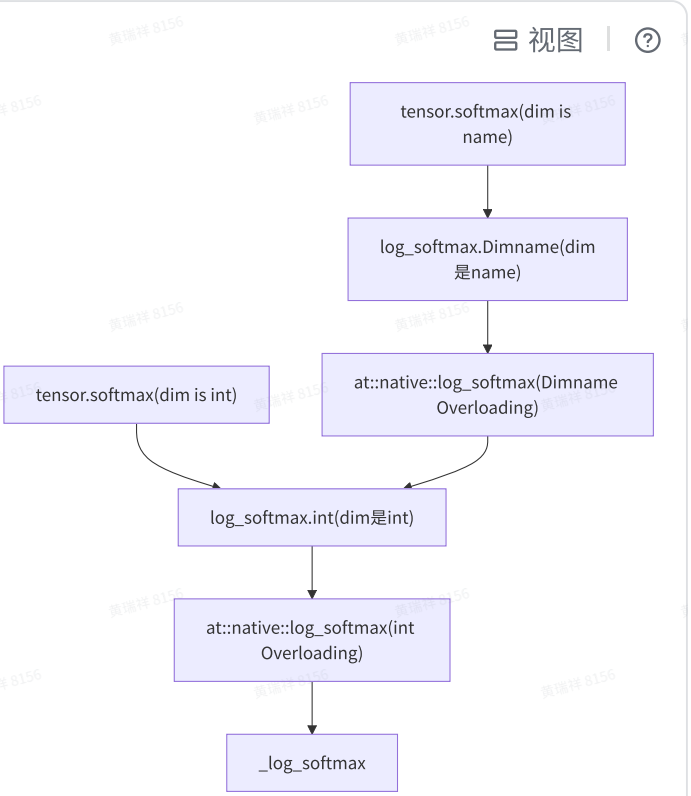
- [ ] Dtype 支持
  - [x] ~~float32~~
  - [ ] float16
  - [ ] bfloat16 目前缺少对应的指令
  - [ ] mxformat
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
- [ ] Memory 是否需要 Align，Align 到多少
- [ ] 目前需要软件将 dim_size padding 到 256 的倍数
  - [ ] 需要优化
- [ ] 后期需要支持任意维度上做 softmax
  a. 提供给算法的原型还是按照下面表格封装，但是第一步先只支持在最 inner 维度做 softmax，原因参考下面的分析。
  b. 第二步视情况考虑实现任意维度 softmax，当前有个想法是在 RVV 上利用 index load/store 来做 gather，将参与计算的元素都收集起来连续存放，不用 tile core 的原因是 tile 的 index load/store 粒度太大，最小也是 32B，而这个场景的需求是按照元素来 index load/store。softmax

log_softmax.int,log_softmax.Dimname,_log_softmax

OP

Pytorch 官方文档参考：
https://pytorch.org/docs/stable/generated/torch.nn.LogSoftmax.html#torch.nn.LogSoftmax

```
flowchart TD
    F["tensor.softmax(dim is int)"] -->
    G["tensor.softmax(dim is name)"] --
    A --> B["at::native::log_softmax(int
    C --> D["at::native::log_softmax(Dim
    D --> A
    B --> E["_log_softmax"]
```



_log_softmax

mathematically equivalent to log(softmax(x))

参考信息

OP List 链接：https://siorigin.feishu.cn/record/GXmArsJOSeXZepcWqO3clgQXnah

Pytorch 官方文档参考：
https://pytorch.org/docs/stable/generated/torch.nn.LogSoftmax.html#torch.nn.LogSoftmax

Function 接口

代码块

```
void fn(dev, out*, input*,  bool half_to_float,  int64_t outer_size, dim_size, inner_size)
//(updated)
void fn(dev, const void* out,const void* self,  bool half_to_float,  int64_t outer_size, dim_size, inner_size)
// 其中dim_size是dim参数所在维度的size，outer size是所在维度小于dim参数的所有维度大小:
```

```
// for (int64_t i = 0; i < dim; ++i)
//       outer_size *= input.size(i);
// 例如对于shape为[A,B,C,D,E]的Tensor在C维度做Softmax，会view成[A]
```

ISSUE

- [ ] Dtype 支持
  - [ ] float32
  - [ ] float16
  - [ ] bfloat16 目前缺少对应的指令
  - [ ] mxformat
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
- [ ] Memory 是否需要 Align，Align 到多少
- [ ] 目前需要软件将 dim_size padding 到 256 的倍数
  - [ ] 需要优化
- [ ] 后期需要支持任意维度上做 softmax
  a. 提供给算法的原型还是按照下面表格封装，但是第一步先只支持在最 inner 维度做 softmax，原因参考下面的分析。
  b. 第二步视情况考虑实现任意维度 softmax，当前有个想法是在 RVV 上利用 index load/store 来做 gather，将参与计算的元素都收集起来连续存放，不用 tile core 的原因是 tile 的 index load/store 粒度太大，最小也是 32B，而这个场景的需求是按照元素来 index load/store。 softmax

_local_scalar_dense

在将 scalar 转换为 cpu scalar 时会调用，应该直接使用 runtime API 就可以实现，参考 CUDA 实现：_local_scalar_dense_cuda

代码块

```
// schema: func: _local_scalar_dense(Tensor self) -> Scalar
```

```
In [12]: torch.randn(1).cuda().item()
 [call] op=[aten::randn], key=[BackendSelect]
  [redispatch] op=[aten::randn], key=[CPU]
   [call] op=[aten::empty.memory_format], key=[BackendSelect]
    [redispatch] op=[aten::empty.memory_format], key=[CPU]
   [call] op=[aten::normal_], key=[CPU]
 [call] op=[aten::to.dtype_layout], key=[AutogradCPU]
  [call] op=[aten::_to_copy], key=[AutogradCPU]
   [redispatch] op=[aten::_to_copy], key=[BackendSelect]
    [redispatch] op=[aten::_to_copy], key=[CUDA]
     [call] op=[aten::empty_strided], key=[BackendSelect]
      [redispatch] op=[aten::empty_strided], key=[CUDA]
     [call] op=[aten::copy_], key=[CUDA]
 [call] op=[aten::item], key=[AutogradCUDA]
  [call] op=[aten::_local_scalar_dense], key=[AutogradCUDA]
   [redispatch] op=[aten::_local_scalar_dense], key=[CUDA]
Out[12]: -0.9774109721183777
```

add.Tensor

参考信息

只能使用 TileCore 实现，需要

按道理应该注册到 CompositeImplicitAutograd，但是却直接注册到了 CUDA；看起来像是针对 tag 为 ufunc_inner_loop 的 op 专门在 gen.py 中做了处理：

```
for g in structured_native_functions:
    if not g.out.ufunc_inner_loop or not is_ufunc_dispatch_key(dispatch_key):
        continue
    name = g.functional.func.name.name
    if dispatch_key is DispatchKey.CPU:
        assert fm is cpu_fm
        fm.write_with_template(
            f"UfuncCPU_{name}.cpp",
            "UfuncCPU.cpp",
            lambda: {
                "meta_declaration": compute_meta_function_declaration(g),
                "native_declaration": dest.compute_native_function_declaration(
                    g, backend_indices[dispatch_key]
                ),
                "native_definitions": dest.compute_ufunc_cpu(g),
            },
        )
        cpu_vec_fm.write_with_template(
            f"UfuncCPUKernel_{name}.cpp",
            "UfuncCPUKernel.cpp",
            lambda: {
                "name": name,
                "native_definitions": dest.compute_ufunc_cpu_kernel(g),
            },
        )
    elif dispatch_key is DispatchKey.CUDA:
        cuda_headers = "#include <ATen/native/cuda/Loops.cuh>"
        if rocm:
            cuda_headers = "#include <ATen/native/hip/Loops.cuh>"
        fm.write_with_template(
            f"UfuncCUDA_{name}.cu",
            "UfuncCUDA.cu",
            lambda: {
                "name": name,
                "cuda_headers": cuda_headers,
                "meta_declaration": compute_meta_function_declaration(g),
                "native_declaration": dest.compute_native_function_declaration(
                    g, backend_indices[dispatch_key]
                ),
                "native_definitions": dest.compute_ufunc_cuda(g),
            },
        )
    else:
        raise AssertionError(f"unrecognized {dispatch_key} for ufunc")
```

参考信息

OP List 链接：https://siorigin.feishu.cn/record/Hy3BrJjkVel59HcGDWycyJ7Wnfg

Pytorch 官方文档参考： https://pytorch.org/docs/stable/generated/torch.add

具体对应的 Op 参考：THPVariable_add

Cuda 实现参考：

- 源代码里没有对应实现，为自动生成，生成后在 build/aten/src/ATen/UfuncCUDA_add.cu 文件
- 直接调用封装好的 gpu_kernel，支持传入一个 Scalar + 一个 Tensor 或者两个 Tensor
- Kernel 需处理 self/other 其中一个为 scalar 的情况
- gpu_kernel 中相关的优化参考 neg.out 中的 *Performance*

Function 接口

代码块

```cpp
// schema: add.out(Tensor self, Tensor other, *, Scalar alpha=1, Tensor(a!) out) -> Tensor(a!)
// schema: add_.Tensor(Tensor(a!) self, Tensor other, *, Scalar alpha=1) -> Tensor(a!)
// schema: add.Tensor(Tensor self, Tensor other, *, Scalar alpha=1) -> Tensor

template<typename T>
void add_out(
    const T* self,
    const int_t self_ndims,
    const int64_t* self_sizes,
    const T* other,
    const int64_t other_ndims,
    const int64_t* other_sizes,
    T alpha = 1,
    T* out,
    const int64_t out_ndims,
    const int64_t* out_sizes,
    const int64_t* element_sizes = nullptr)
```

ISSUE

- [ ] Dtype 支持
  - [ ] bfloat16 目前缺少对应的指令
  - [ ] Mxformat
  - [ ] `uint8_t, int8_t, int16_t, int, int64_t, float, double, c10::complex<float>, c10::complex<double>, c10::Half, bool, at::BFloat16`, `c10::complex<c10::Half>`
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
- [ ] Memory 是否需要 Align，Align 到多少
- [ ] 目前需要软件 pad 到 tile format
  - [ ] 需要优化
- [ ] 目前对于需要 broadcast 的情况，软件来实现 expand
  - [ ] 对于 outer dim broadcast 需要优化

bmm,bmm.out

参考信息

Pytorch 官方文档参考： https://pytorch.org/docs/stable/generated/torch.bmm

具体对应的 Op 参考：THPVariable_bmm

Function 接口

代码块

```
// schema: bmm(Tensor self, Tensor mat2) -> Tensor
// schema: bmm.out(Tensor self, Tensor mat2, *, Tensor(a!) out) -> Tensor(a!)

// 实现接口参考：https://docs.nvidia.com/cuda/cublas/#cublasgemmstridedbatchedex
```

any.all_out, any,any.dims,any.dim,any.out,any.dimname

对应 pytorch 接口： https://pytorch.org/docs/stable/generated/torch.any.html，具体对应的 Op 参考 THPVariable_any

参考实现：

https://github.com/pytorch/pytorch/blob/3f069e7679588d5ee4b1d5b2492ca0e20f9320b5/aten/src/ATen/native/ReduceOps.cpp#L1611

cuda 中 any 和 all 是放在一起实现的

代码块

```
// 1. 基本用法 - 全局规约
{
    // any - 任意元素为True则返回True
    auto input = torch::tensor({1, 0, 1});
    auto result = torch::any(input);  // tensor(True)

    // all - 所有元素为True才返回True
    input = torch::tensor({1, 1, 1});
    result = torch::all(input);  // tensor(True)
}

// 2. 指定维度规约
{
    // 2x3 tensor
    auto input = torch::tensor({{1, 0, 1},
                                {0, 1, 1}});

    // any 在维度0上规约
    auto result = torch::any(input, 0);  // [1, 1, 1]

    // all 在维度1上规约
    result = torch::all(input, 1);  // [0, 0]
}

// 3. 保持维度 (keepdim=true)
```

```cpp
{
    auto input = torch::tensor({{1, 0, 1},
                                {0, 1, 1}});

    // any 在维度0上规约，保持维度
    auto result = torch::any(input, 0, true);  // [[1, 1, 1]]

    // all 在维度1上规约，保持维度
    result = torch::all(input, 1, true);  // [[0], [0]]
}

// 4. uint8类型处理
{
    auto input = torch::tensor({1, 0, 1}, torch::kUInt8);

    // 对uint8类型，输出保持uint8
    auto result = torch::any(input);  // tensor(1, dtype=uint8)
    result = torch::all(input);  // tensor(0, dtype=uint8)
}

// 5. 使用out参数
{
    auto input = torch::tensor({1, 0, 1});
    auto out = torch::empty({}, torch::kBool);

    // any with out
    torch::any_out(out, input);

    // all with out
    torch::all_out(out, input);
}

// 6. 多维度规约
{
    // 2x3x2 tensor
    auto input = torch::ones({2, 3, 2});
    std::vector<int64_t> dims = {0, 2};

    // 在维度0和2上规约
    auto result = torch::any(input, dims);  // [1, 1, 1]
    result = torch::all(input, dims);  // [1, 1, 1]
}

// 7. 空张量处理
{
    auto input = torch::tensor({});

```

```
    // 空张量的any返回false
    auto result = torch::any(input);  // tensor(False)

    // 空张量的all返回true
    result = torch::all(input);  // tensor(True)
}
```

any.all_out,any

CUDA 参考实现 any_all_out

代码块

```
// schema: any.all_out(Tensor self, *, Tensor(a!) out) -> Tensor(a!)
// schema: all(Tensor self) -> Tensor
```

```
tensor([[ 0.6800, -1.9964,  0.2954],
        [-1.5758, -0.9457,  1.0380],
        [-1.4764,  0.5204, -0.0701]], device='cuda:0')

In [32]: a.any()
 [call] op=[aten::any], key=[AutogradCUDA]
  [redispatch] op=[aten::any], key=[CUDA]
   [call] op=[aten::as_strided], key=[CUDA]
Out[32]:  [call] op=[aten::reshape], key=[AutogradCUDA]
  [call] op=[aten::view], key=[AutogradCUDA]
   [redispatch] op=[aten::view], key=[ADInplaceOrView]
    [redispatch] op=[aten::view], key=[CUDA]
 [call] op=[aten::unbind.int], key=[AutogradCUDA]
  [redispatch] op=[aten::unbind.int], key=[ADInplaceOrView]
   [redispatch] op=[aten::unbind.int], key=[CUDA]
    [call] op=[aten::select.int], key=[CUDA]
     [call] op=[aten::as_strided], key=[CUDA]
 [call] op=[aten::item], key=[AutogradCUDA]
  [call] op=[aten::_local_scalar_dense], key=[AutogradCUDA]
   [redispatch] op=[aten::_local_scalar_dense], key=[CUDA]
 [call] op=[aten::item], key=[AutogradCUDA]
  [call] op=[aten::_local_scalar_dense], key=[AutogradCUDA]
   [redispatch] op=[aten::_local_scalar_dense], key=[CUDA]
tensor(True, device='cuda:0')
```

any.dim,any.out

CUDA 参考实现 any_out

代码块

```
// schema: any.dim(Tensor self, int dim, bool keepdim=False) -> Tensor
// schema: any.out(Tensor self, int dim, bool keepdim=False, *, Tensor(a!) out) -> Tensor(a!)
```

```
In [45]: a.shape
Out[45]: torch.Size([2, 3, 4])

In [46]: a.any(dim=0)
 [call] op=[aten::any.dim], key=[AutogradCUDA]
  [redispatch] op=[aten::any.dim], key=[CUDA]
   [call] op=[aten::as_strided], key=[CUDA]
```

```
tensor([[True, True, True, True],
        [True, True, True, True],
        [True, True, True, True]], device='cuda:0')
```

all.dimname, all.dimname_out, all.dims, all.dims_out,

官方文档

https://pytorch.org/docs/stable/generated/torch.all.html

代码块

```
//all.dimname_out(Tensor self, Dimname dim, bool keepdim=False, *, Tensor(a!) out) -> Tensor(a!)
//all.dimname(Tensor self, Dimname dim, bool keepdim=False) -> Tensor
//all.dims(Tensor self, int[]? dim=None, bool keepdim=False) -> Tensor
//all.dims_out(Tensor self, int[]? dim=None, bool keepdim=False, *, Tensor(a!) out) -> Tensor(a!)


```

all.dim, all.out

代码块

```
//all.dims(Tensor self, int[]? dim=None, bool keepdim=False) -> Tensor
//all.dims_out(Tensor self, int[]? dim=None, bool keepdim=False, *, Tensor(a!) out) -> Tensor(a!)
```

Function 接口

代码块

```
/**
 * @brief Compute logical any/all reduction over dimensions of input tensor
 *
 * @param dev           Device handle for CUDA device management
 * @param out          Output tensor (pre-allocated, bool or uint8)
 * @param self         Input tensor
 * @param dims         Dimensions to reduce over (optional)
 * @param keepdim      Whether the output tensor has dim retained or not
 * @param dtype        Data type of input tensor
 * @param is_all       If true compute logical AND, else compute logical OR
 *
 * @note Output Type:
 *       - For uint8 input: output is uint8
 *       - For all other inputs: output is bool
 *
 * @note Dimension Rules:
 *       - If dim is empty, reduce over all dimensions
 *       - Multiple dims are allowed
 *       - Negative dims are wrapped
 *
 * @note Examples:
 *       any: [1,1,0] -> true
 *       any: [0,0,0] -> false
 *       all: [1,1,0] -> false
 *       all: [1,1,1] -> true
```

```
 *
 * @return int 0 on success, error code otherwise
 */
void any_all_fn(
    SiDeviceHandle dev,          // Device handle
    void *out,                   // Output tensor (bool or uint8)
    const void *self,            // Input tensor
    const int64_t* dims,         // Dimensions to reduce over (optional)
    bool keepdim,                // Whether to keep reduced dimensions
    ScalarType dtype,            // Input data type
    bool is_all                  // true for all(), false for any()
);
```

bitwise_and.Tensor,bitwise_and.Tensor_out,bitwise_and_.Tensor,bitwise_and.Scalar_out,bitwise_and.Scalar,bitwise_and.Scalar_Tensor,bitwise_and.Scalar_Tensor_out,bitwise_and_.Scalar

参考信息

只能使用 RVV 实现

Pytorch 官方文档参考：https://pytorch.org/docs/stable/generated/torch.bitwise_and.html

具体对应的 Op 参考：THPVariable_bitwise_and_，THPVariable_bitwise_and

支持 broadcastable 的 tensor 计算

对于 int 类型的 tensor，运算后的结果是二进制补码对应的十进制数

在线补码网站：https://www.23bei.com/tool/56.html?

bitwise_and.Tensor,bitwise_and.Tensor_out,bitwise_and_.Tensor

CUDA 实现搜索：bitwise_and_kernel_cuda

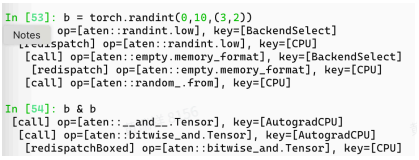
Function 接口

建议在 device 和 kernel 可以联合编译后参考 CUDA 代码来实现

代码块

```
// schema: bitwise_and.Tensor(Tensor self, Tensor other) -> Tensor
// schema: bitwise_and_.Tensor(Tensor(a!) self, Tensor other) -> Tensor(a!)
// schema: bitwise_and.Tensor_out(Tensor self, Tensor other, *, Tensor(a!) out) -> Tensor(a!)

/**
nargs: num of output+input tensors
```



```
7   args: output+input tensors' ptr
8   dims: output tensor dims
9   sizes: output tensor sizes
10  strides: strides of nargs tensors, 0 means it's a
    broadcasted dim
11  element_sizes: element_size of nargs tensor; if
    element_sizes is nullptr, then the strides will
    be in bytes, otherwise the strides will be in #
    of elements.
12   **/
13  void fn(dev, int nargs, const void* const* args,
    int dims, const int64_t* sizes, const int64_t*
    const* strides, const int64_t*
    element_sizes=nullptr)
14
15  // 可选的形式
16  void fnv2(dev, const void* out, const void* a,
    const void* b, int dims, const int64_t* sizes,
    const int64_t* strides_out, const int64_t*
    strides_a, const int64_t* strides_b, const
    int64_t* element_sizes=nullptr)
17  //updated
18  void fnv2(dev, const void* out, const void* self,
    const void* other, int out_ndims, const int64_t*
    out_sizes, const int64_t* out_strides, const
    int64_t* self_strides, const int64_t*
    other_strides, const int64_t*
    element_sizes=nullptr)
```

ISSUE

- [ ] Dtype 支持
  - [ ] float32
  - [ ] float16
  - [ ] bfloat16 目前缺少对应的指令
  - [ ] mxformat
- [ ] 非 contiguous 的是否能支持
- [ ] Memory 是否需要 Align，Align 到多少

bitwise_and.Scalar_out

通过 at::native::bitwise_and_out 直接将 scalar 转换为 tensor 的方式，调用 tensor 的实现版本

```
Tensor& bitwise_and_out(const Tensor& self, const Scalar& other, Tensor& result) {
  return at::bitwise_and_out(result, self, wrapped_scalar_tensor(other));
}
```

bitwise_and.Scalar,bitwise_and.Scalar_Tensor,bitwise_and.Scalar_Tensor_out

通过 at::native::bitwise_and 直接将 scalar 转换为 tensor 的方式，调用 tensor 的实现版本

```
Tensor bitwise_and(const Tensor& self, const Scalar& other) {
  return at::bitwise_and(self, wrapped_scalar_tensor(other));
}

Tensor bitwise_and(const Scalar& self, const Tensor& other) {
  return at::bitwise_and(wrapped_scalar_tensor(self), other);
}
```

bitwise_and_.Scalar

通过 at::native::bitwise_and_直接将函数调用转换为 tensor.bitwise_and_一个 tensor 的方式实现

```
Tensor& bitwise_and_(Tensor& self, const Scalar& other) {
  return self.bitwise_and_(wrapped_scalar_tensor(other));
}
```

bitwise_not,bitwise_not_,bitwise_not.out

参考信息

只能使用 RVV 实现

https://pytorch.org/docs/stable/generated/torch.bitwise_not

具体对应的 Op 参考 THPVariable_bitwise_not

对于 int 类型的 tensor，运算后的结果是二进制补码对应的十进制数，每个 tensor 中的数单独按照对小对应的 8bit/16bit/32bit/64bit 来计算，例如对 a = tensor（[1， 65536]）， ～a = tensor（[-2， -65537]）

在线补码网站：https://www.23bei.com/tool/56.html?

Function 接口

建议在 device 和 kernel 可以联合编译后参考 CUDA 代码来实现

代码块

```
1  // schema: bitwise_not(Tensor self) -> Tensor
```

```
tensor([[8, 5],
        [4, 8],
        [6, 6]])

In [91]: (b, ~b)
 [call] op=[aten::bitwise_not], key=[AutogradCPU]
  [redispatchBoxed] op=[aten::bitwise_not], key=[CPU]
```

```
// schema: bitwise_not_(Tensor(a!) self) -> Tensor(a!)
// schema: bitwise_not.out(Tensor self, *, Tensor(a!) out) -> Tensor(a!)

/**
out: output tensor
a: input tensor
dims: output tensor dims
sizes: output tensor sizes
strides: strides of output tensor
element_sizes: element_size of nargs tensor; if element_sizes is nullptr, then the strides will be in bytes, otherwise the strides will be in # of elements.
 **/
void fn(dev, const void* out, const void* a, int dims, const int64_t* sizes, const int64_t* strides, const int64_t* element_sizes=nullptr)
//updated
void fn(dev, const void* out, const void* self, int out_ndims, const int64_t* out_sizes, const int64_t* out_strides, const int64_t* element_sizes=nullptr)
```

```
(tensor([[8, 5],
         [4, 8],
         [6, 6]]),
 tensor([[-9, -6],
         [-5, -9],
         [-7, -7]]))
```

ISSUE

- [ ] Dtype 支持
  - [ ] float32
  - [ ] float16
  - [ ] bfloat16 目前缺少对应的指令
  - [ ] mxformat
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
- [ ] Memory 是否需要 Align，Align 到多少

bitwise_or.Tensor,bitwise_or_.Tensor,bitwise_or.Tensor_out

参考信息

只能使用 RVV 实现

https://pytorch.org/docs/stable/generated/torch.bitwise_or

支持 broadcastable 的 tensor 计算

具体对应的 Op 参考 THPVariable_bitwise_orq

代码块

```
// schema: bitwise_or.Tensor(Tensor self, Tensor other) -> Tensor
// schema: bitwise_or_.Tensor(Tensor(a!) self, Tensor other) -> Tensor(a!)
// schema: bitwise_or.Tensor_out(Tensor self, Tensor other, *, Tensor(a!) out) -> Tensor(a!)
```

Function 接口

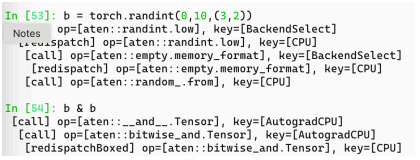
建议在 device 和 kernel 可以联合编译后参考 CUDA 代码来实现

代码块

```
// schema: bitwise_and.Tensor(Tensor self, Tensor other) -> Tensor
// schema: bitwise_and_.Tensor(Tensor(a!) self, Tensor other) -> Tensor(a!)
// schema: bitwise_and.Tensor_out(Tensor self, Tensor other, *, Tensor(a!) out) -> Tensor(a!)

/**
nargs: num of output+input tensors
args: output+input tensors' ptr
dims: output tensor dims
sizes: output tensor sizes
strides: strides of nargs tensors, 0 means it's a broadcasted dim
element_sizes: element_size of nargs tensor; if element_sizes is nullptr, then the strides will be in bytes, otherwise the strides will be in # of elements.
 **/
void fn(dev, int nargs, const void* const* args, int dims, const int64_t* sizes, const int64_t* const* strides, const int64_t* element_sizes=nullptr)

// 可选的形式
void fnv2(dev, const void* out, const void* a, const void* b, int dims, const int64_t* sizes, const int64_t* strides_out, const int64_t*
```



```
    strides_a, const int64_t* strides_b, const
    int64_t* element_sizes=nullptr)
17  //updated
18  void fnv2(dev, const void* out, const void* self,
    const void* other, int out_ndims, const int64_t*
    out_sizes, const int64_t* out_strides, const
    int64_t* self_strides, const int64_t*
    other_strides, const int64_t*
    element_sizes=nullptr)
```

ISSUE

- [ ] Dtype 支持
  - [ ] float32
  - [ ] float16
  - [ ] bfloat16 目前缺少对应的指令
  - [ ] mxformat
- [ ] 非 contiguous 的是否能支持
- [ ] Memory 是否需要 Align，Align 到多少

eq.Tensor,eq.Tensor_out,eq_.Tensor

参考信息

只能使用 RVV 实现

https://pytorch.org/docs/stable/generated/torch.eq

支持 broadcastable 的 tensor 计算

具体对应的 Op 参考 THPVariable_eq

代码块

```
1  // schema: eq_.Tensor(Tensor(a!) self,
   Tensor other) -> Tensor(a!)
2  // schema: eq.Tensor_out(Tensor self,
   Tensor other, *, Tensor(a!) out) ->
   Tensor(a!)
3  // schema: eq.Tensor(Tensor self, Tensor
   other) -> Tensor
```

```
In [165]: (c, d, c==d)
 [call] op=[aten::eq.Tensor], key=[AutogradCPU]
  [redispatch] op=[aten::eq.Tensor], key=[CPU]
```

```
(tensor([[    1, 65536],
         [    2,     3]]),
 tensor([   22, 65536]),
 tensor([[False,  True],
         [False, False]]))
```

Function 接口

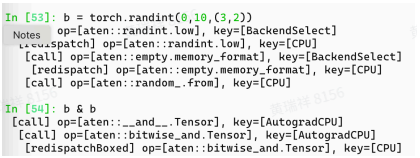
建议在 device 和 kernel 可以联合编译后参考 CUDA 代码来实现

代码块

```
// schema: bitwise_and.Tensor(Tensor self, Tensor other) -> Tensor
// schema: bitwise_and_.Tensor(Tensor(a!) self, Tensor other) -> Tensor(a!)
// schema: bitwise_and.Tensor_out(Tensor self, Tensor other, *, Tensor(a!) out) -> Tensor(a!)

/**
nargs: num of output+input tensors
args: output+input tensors' ptr
dims: output tensor dims
sizes: output tensor sizes
strides: strides of nargs tensors, 0 means it's a broadcasted dim
element_sizes: element_size of nargs tensor; if element_sizes is nullptr, then the strides will be in bytes, otherwise the strides will be in # of elements.
 **/
void fn(dev, int nargs, const void* const* args, int dims, const int64_t* sizes, const int64_t* const* strides, const int64_t* element_sizes=nullptr)

// 可选的形式
void fnv2(dev, const void* out, const void* a, const void* b, int dims, const int64_t* sizes, const int64_t* strides_out, const int64_t* strides_a, const int64_t* strides_b, const int64_t* element_sizes=nullptr)
// updated
void fnv2(dev, const void* out, const void* self, const void* other, int out_ndim, const int64_t* out_sizes, const int64_t* out_strides, const int64_t* self_strides, const int64_t* other_strides, const int64_t* element_sizes=nullptr)
```



首个版本开发说明

（1）数制支持上，需要支持类型有：int32_t，bfloat16，float32

（2）初版只需要支持输入2为scalar的情况

（3）由于硬件限制，对于int64_t类型，由框架将数据进行64-32转换

内部接口调用形式：

代码块

```
/// @brief Computes element-wise equality
/// @tparam T Datatype support [int32_t, sifmt:float32, sifmt::bfloat16]
/// @param[in] d_in0 Input tensor 0 with shape [dim0_size, dim1_size]
/// @param[in] d_in1 Input tensor 1 with shape [dim0_size, dim1_size] OR scalar
/// @param[out] d_out Output tensor with shape [dim0_size, dim1_size]
/// @param[in] dim0_size Tensor shape param 0
/// @param[in] dim1_size Tensor shape param 1
/// @param[in] is_d_in1_scalar Indicates tensor 1 is scalar OR tensor
/// @param[in] op_type Indicates OP is EQUAL(0) OR NOT EQUAL(1)
template <typename T>
void eq(void *d_in0, void *d_in1, void *d_out, int dim0_size, int dim1_size,
bool is_d_in1_scalar = true, int op_type = 0);
```

ISSUE

- [ ] Dtype 支持
  - [ ] float32
  - [ ] float16
  - [ ] bfloat16 目前缺少对应的指令
  - [ ] mxformat
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
- [ ] Memory 是否需要 Align，Align 到多少

ge.Scalar,ge.Scalar_out,ge_.Scalar,ge.Tensor,ge.Tensor_out,ge_.Tensor

参考信息

只能使用 RVV 实现

https://pytorch.org/docs/stable/generated/torch.ge

支持 broadcastable 的 tensor 计算

具体对应的 Op 参考 THPVariable_ge

ge.Scalar,ge.Scalar_out,ge_.Scalar

```
In [174]: (c, c > 2)
 [call] op=[aten::gt.Scalar], key=[AutogradCPU]
  [redispatch] op=[aten::gt.Scalar], key=[CPU]
```

代码块

```
// schema: ge.Scalar(Tensor self, Scalar other) -> Tensor
```

```
(tensor([[    1, 65536],
         [    2,     3]]),
 tensor([[False,  True],
         [False,  True]]))
```

```
2   // schema: ge.Scalar_out(Tensor self,
    Scalar other, *, Tensor(a!) out) ->
    Tensor(a!)
3   // schema: ge_.Scalar(Tensor(a!) self,
    Scalar other) -> Tensor(a!)
```

ge.Tensor,ge.Tensor_out,ge_.Tensor

代码块

```
1   // schema: ge_.Tensor(Tensor(a!) self,
    Tensor other) -> Tensor(a!)
2   // schema: ge.Tensor_out(Tensor self,
    Tensor other, *, Tensor(a!) out) ->
    Tensor(a!)
3   // schema: ge.Tensor(Tensor self, Tensor
    other) -> Tensor
```

```
In [179]: (c, d, c>d)
 [call] op=[aten::gt.Tensor], key=[AutogradCUDA]
  [redispatch] op=[aten::gt.Tensor], key=[CUDA]
```

```
(tensor([[    1, 65536],
         [    2,     3]], device='cuda:0'),
 tensor([   22, 65536], device='cuda:0'),
 tensor([[False, False],
         [False, False]], device='cuda:0'))
```

Function 接口

建议在 device 和 kernel 可以联合编译后参考 CUDA 代码来实现

代码块

```
1   // schema: bitwise_and.Tensor(Tensor self, Tensor
    other) -> Tensor
2   // schema: bitwise_and_.Tensor(Tensor(a!) self,
    Tensor other) -> Tensor(a!)
3   // schema: bitwise_and.Tensor_out(Tensor self,
    Tensor other, *, Tensor(a!) out) -> Tensor(a!)
4
5   /**
6   nargs: num of output+input tensors
7   args: output+input tensors' ptr
8   dims: output tensor dims
9   sizes: output tensor sizes
10  strides: strides of nargs tensors, 0 means it's a
    broadcasted dim
11  element_sizes: element_size of nargs tensor; if
    element_sizes is nullptr, then the strides will
    be in bytes, otherwise the strides will be in #
    of elements.
12   **/
13  void fn(dev, int nargs, const void* const* args,
    int dims, const int64_t* sizes, const int64_t*
```

```
In [53]: b = torch.randint(0,10,(3,2))
Notes
    op=[aten::randint.low], key=[BackendSelect]
  [redispatch] op=[aten::randint.low], key=[CPU]
   [call] op=[aten::empty.memory_format], key=[BackendSelect]
    [redispatch] op=[aten::empty.memory_format], key=[CPU]
   [call] op=[aten::random_.from], key=[CPU]

In [54]: b & b
 [call] op=[aten::__and__.Tensor], key=[AutogradCPU]
  [call] op=[aten::bitwise_and.Tensor], key=[AutogradCPU]
   [redispatchBoxed] op=[aten::bitwise_and.Tensor], key=[CPU]
```

```
    const* strides, const int64_t*
    element_sizes=nullptr)

// 可选的形式
void fnv2(dev, const void* out, const void* a,
const void* b, int dims, const int64_t* sizes,
const int64_t* strides_out, const int64_t*
strides_a, const int64_t* strides_b, const
int64_t* element_sizes=nullptr)
// updated
void fnv2(dev, const void* out, const void* self,
const void* other, int out_ndims, const int64_t*
out_sizes, const int64_t* out_strides, const
int64_t* self_strides, const int64_t*
other_strides, const int64_t*
element_sizes=nullptr)
```

ISSUE

- [ ] Dtype 支持
  - [ ] float32
  - [ ] float16
  - [ ] bfloat16 目前缺少对应的指令
  - [ ] mxformat
- [ ] 非 contiguous 的是否能支持
- [ ] Memory 是否需要 Align，Align 到多少

mul.Tensor, mul_.Tensor, mul.out

参考信息

Pytorch 官方文档参考：https://pytorch.org/docs/stable/generated/torch.mul.html

cuda 实现参考：

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/BinaryMulKernel.cu
- 直接调用封装好的 gpu_kernel，支持传入一个 Scalar + 一个 Tensor 或者两个 Tensor
- Kernel 需处理 self/other 其中一个为 scalar 的情况
- gpu_kernel 中相关的优化参考 neg.out 中的 *Performance*

Function 接口

代码块

```
// schema: mul.out(Tensor self, Tensor other, *, Tensor(a!) out) -> Tensor(a!)
// schema: mul.Tensor(Tensor self, Tensor other) -> Tensor
// schema: mul_.Tensor(Tensor(a!) self, Tensor other) -> Tensor(a!)

template<typename T>
void mul_out(
    const T* self,
    const int64_t self_ndims,
    const int64_t* self_sizes,
    const T* other,
    const int64_t other_ndims,
    const int64_t* other_sizes,
    T* out,
    const int64_t out_ndims,
    const int64_t* out_sizes,
    const int64_t* element_sizes = nullptr)
```

ISSUE

- [ ] Dtype 支持
  - [ ] `uint8_t, int8_t, int16_t, int, int64_t, float, double, c10::complex<float>, c10::complex<double>, c10::Half, bool, at::BFloat16`, `c10::complex<c10::Half>`

gather,gather.out

参考信息

OP List 链接：https://siorigin.feishu.cn/record/L6XorIxSxeO1g8c8DdFcqnUtnbd

Pytorch 官方文档参考：https://pytorch.org/docs/stable/generated/torch.gather.html

> Output tensor 的 shape 和 index 的 tensor shape 一致

实现参考：

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/ScatterGatherKernel.cu#L219
- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/ScatterGatherKernel.cu#L117

Function 接口

代码块

```
// schema: gather.out(Tensor self, int dim, Tensor index, *, bool
sparse_grad=False, Tensor(a!) out) -> Tensor(a!)
```

```
// schema: gather(Tensor self, int dim, Tensor index, *, bool sparse_grad=False) -> Tensor

/**
 * @brief Performs a gather operation on the input tensor `self` using indices from the `index` tensor, and stores the result in the output tensor `out`.
 *
 * @param dev             Device on which the operation is performed.
 * @param out             Pointer to the output tensor, which is modified in place.
 * @param dims            Common number of dimensions shared by `out`, `self`, and `index`.
 * @param out_sizes       Sizes of each dimension for both the output tensor and the index tensor.
 * @param out_strides     Strides of each dimension of the output tensor.
 * @param self            Pointer to the input tensor.
 * @param self_sizes      Sizes of each dimension of the input tensor.
 * @param self_strides    Strides of each dimension of the input tensor.
 * @param dim             Dimension along which elements are gathered from `self`.
 * @param index           Pointer to the index tensor, which is a multi-dimensional tensor containing indices.
 * @param index_stride    Strides of each dimension of the index tensor.
 * @param element_sizes   Optional pointer to the element sizes of each tensor (if nullptr, strides are in bytes).
 */
void fn(dev, void* out, int dims, const int64_t* out_sizes, const int64_t* out_strides, const void* self, const int64_t* self_sizes, const int64_t* self_strides, int64_t dim, const void* index, const int64_t* index_stride, const int64_t* element_sizes=nullptr)
//updated
void fn(dev, void* out, int ndims, const int64_t* out_sizes, const int64_t* out_strides, const void* self, const int64_t* self_sizes, const int64_t* self_strides, int64_t dim, const void* index, const int64_t* index_strides, const int64_t* element_sizes=nullptr)
```

一些例子

代码块

```
# 参考pytorch文档https://pytorch.org/docs/stable/generated/torch.gather.html，这里index=torch.tensor([[0, 0], [1, 0]]), shape为[2,2], 所以ij分别为
# 0,0; 0,1; 1,0; 1,1
t = torch.tensor([[1, 2], [3, 4]])
torch.gather(t, 1, torch.tensor([[0, 0], [1, 0]]))

```

```
# Out[33]:
# tensor([[1, 1],
#         [4, 3]])

# 这里index=torch.tensor([[[1,1]],[[0,1]]]), shape为[2,1,2], 所以ijk分别为
# 0,0,0; 0,0,1; 1,0,0; 1,0,1
t = torch.arange(8).reshape(2,2,2)
torch.gather(t, 1, torch.tensor([[[1,1]],[[0,1]]]))

# Out[32]:
# tensor([[[2, 3]],
#
#         [[4, 7]]])
```

首个版本开发说明

（1）支持输入输出2D tensor的数据为float32类型，index内的数据类型为int64。

（2）dim默认为-1，可支持0/1。

（3）维度信息包括input_dim0,input_dim1,index_dim0,index_dim1。output维度与index相同，不需要提供。

内部接口调用形式：

代码块

```
template <typename T>
void gather(void *d_in, void *d_index, void *d_out, int dim=-1,
                                                    int in_dim0, int in_dim1,
                                                    int index_dim0, int
index_dim1);
```

ISSUE

- [ ] Dtype 支持
  - AT_DISPATCH_ALL_TYPES_AND_COMPLEX: `uint8_t, int8_t, int16_t, int, int64_t, float, double, c10::complex<float>, c10::complex<double>`
  - `c10::Half, bool, at::BFloat16`
  - MX Format?

index_select

参考信息

OP List 链接：https://siorigin.feishu.cn/record/XkXcrPomqe6yO0cma8XclHYjnlf

Pytorch 官方文档参考：https://pytorch.org/docs/stable/generated/torch.index_select.html

实现参考：

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/Indexing.cu#L1350
- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/Indexing.cu#L1240
- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/Indexing.cu#L1283

example

代码块

```
>>> x = torch.randn(3, 4)
>>> x
tensor([[ 1.1880,  0.2477, -1.3225,  1.4690],
        [ 2.2300, -0.1451, -0.1112, -0.5504],
        [ 0.1272, -0.8875,  0.2266, -0.3423]])
>>> indices = torch.tensor([0, 2])
>>> res1 = torch.index_select(x, 0, indices)
>>> res1
tensor([[ 1.1880,  0.2477, -1.3225,  1.4690],
        [ 0.1272, -0.8875,  0.2266, -0.3423]])
>>> res2 = torch.index_select(x, 1, indices)
>>> res2
tensor([[ 1.1880, -1.3225],
        [ 2.2300, -0.1112],
        [ 0.1272,  0.2266]])
```

Function 接口

代码块

```
// schema: index_select(Tensor self, int dim, Tensor index) -> Tensor

/**
 * @brief Performs an index_select operation on the input tensor `self` using the provided `index` tensor, and stores the result in the output tensor `out`.
 *
 * @param dev            Device on which the operation is performed.
 * @param out            Pointer to the output tensor.
 * @param dims           Number of dimensions in the input/output tensor.
 * @param out_sizes      Sizes of each dimension of the output tensor.
 * @param out_strides    Strides of each dimension of the output tensor.
```

```
 * @param self          Pointer to the input tensor.
 * @param self_sizes    Sizes of each dimension of the input tensor.
 * @param self_strides  Strides of each dimension of the input tensor.
 * @param dim           Dimension along which index_select is performed.
 * @param index         Pointer to the one-dimensional tensor containing
indices for indexing.
 * @param index_size    Number of elements in the index tensor.
 * @param index_stride  Stride of the index tensor.
 * @param element_sizes Optional pointer to the element sizes of each tensor
(if nullptr, strides are in bytes).
 */
void fn(dev, void* out, int dims, const int64_t* out_sizes, const int64_t*
out_strides, const void* self, const int64_t* self_sizes, const int64_t*
self_strides, int64_t dim, const void* index, const int64_t index_size, const
int64_t index_stride, const int64_t* element_sizes=nullptr)
//updated
void fn(dev, void* out, int ndims, const int64_t* out_sizes, const int64_t*
out_strides, const void* self, const int64_t* self_sizes, const int64_t*
self_strides, int64_t dim, const void* index, const int64_t index_sizes, const
int64_t index_strides, const int64_t* element_sizes=nullptr)
```

ISSUE

- [ ] Dtype 支持
  - AT_ALL_TYPES_AND_COMPLEX: `uint8_t, int8_t, int16_t, int, int64_t, float, double, c10::complex<float>, c10::complex<double>`
  - AT_BAREBONES_UNSIGNED_TYPES: `uint16_t, uint32_t, uint64_t`
  - `c10::complex<c10::Half>, c10::Half, bool, at::BFloat16`
  - MX Format?

silu

unary op， 接口形式同 neg，输入输出 shape 一致

官方参考文档

- https://pytorch.org/docs/stable/generated/torch.nn.functional.silu.html

cuda kernel 参考实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/ActivationSiluKernel.cu#L23

exp 计算需用到 sfu？会对接口形式有影响吗？

```
代码块
```

```
Tensor silu(const Tensor & self); // {"schema": "aten::silu(Tensor self) -> Tensor", "dispatch": "True", "default": "True"}
Tensor & silu_(Tensor & self); // {"schema": "aten::silu_(Tensor(a!) self) -> Tensor(a!)", "dispatch": "True", "default": "True"}
Tensor & silu_out(const Tensor & self, Tensor & out); // {"schema": "aten::silu.out(Tensor self, *, Tensor(a!) out) -> Tensor(a!)", "dispatch": "True", "default": "False"}
```

Dtypes

- AT_FLOATING_TYPES: `float, double`
- AT_COMPLEX_TYPES: `c10::complex<float>, c10::complex<double>`
- Half, BFloat16: `float16, bfloat16`

代码块

```
AT_DISPATCH_FLOATING_AND_COMPLEX_TYPES_AND2(
        at::ScalarType::Half,
        at::ScalarType::BFloat16
```

max, max.dim, max.dim_max

unary op， 接口形式同 neg，输出 size 为 1

官方参考文档

- 注意 aten::max 需要实现的只是不接收 keep_dim / dim 版本的，只是单纯的 max_all
- https://pytorch.org/docs/main/generated/torch.max.html

cuda kernel 参考实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/ReduceMaxValuesKernel.cu#L28
- cuda 实现会封装一个通用的 gpu_reduce_kernel，支持多种 reduce 操作，建议实现类似接口，其他 op 即可通过统一接口接入
- 不同 reduce op 只需传递不同的 ops（即具体的 reduce 函数，如 max 为比较函数），ident（reduce 函数的默认值，如 max 为 scalar_t 类型的最小值）
- 通用接口支持传入要 reduce 的 dim 以及是否 keep_dim

代码块

```
template <typename scalar_t, typename out_scalar_t, int vt0=4, typename ops_t, typename ident_t=double>
```

```
inline void gpu_reduce_kernel(TensorIterator& iter, const ops_t& ops, ident_t ident=0,
                              AccumulationBuffer* acc_buf_ptr=nullptr, int64_t base_idx=0)
```

代码块

```
Tensor max(const Tensor & self); // {"schema": "aten::max(Tensor self) -> Tensor", "dispatch": "True", "default": "False"}


max.dim, max.dim_max用法
>>> x = torch.tensor([[1, 2, 3], [4, 5, 6]])
>>> values = torch.empty(2, 1, dtype=torch.float32)
>>> indices = torch.empty(2, 1, dtype=torch.long)
>>> max_values, max_indices = torch.max(x, dim=1, keepdim=True)
>>> max_values
tensor([[3],
        [6]])
>>> max_indices
tensor([[2],
        [2]])

>>> torch.max(x, dim=1, keepdim=True, out = (max_indices, max_values))
torch.return_types.max_out(
values=tensor([[3],
        [6]]),
indices=tensor([[2],
        [2]]))
```

Dtypes

- AT_ALL_TYPES: `uint8_t, int8_t, int16_t, int, int64_t, float, double`
- Half, Bool, BFloat16: `float16, bool, bfloat16`

代码块

```
AT_DISPATCH_ALL_TYPES_AND3(
      kBFloat16, kHalf, kBool
```

maximum, maximum.out

Binary op, element-wise。两个tensor逐元素比较大小并返回。torch.max(intput, other)会调用torch.maximum接口。

官方文档参考

- https://docs.pytorch.org/docs/stable/generated/torch.maximum.html#torch.maximum

接口（实现maximum_out即可，maximum继承于maximum_out）

```
代码块
1  Tensor maximum(const Tensor & self, const Tensor & other); // {"schema":
   "aten::maximum(Tensor self, Tensor other) -> Tensor", "dispatch": "True",
   "default": "True"}
2  Tensor & maximum_out(const Tensor & self, const Tensor & other, Tensor & out);
   // {"schema": "aten::maximum.out(Tensor self, Tensor other, *, Tensor(a!) out)
   -> Tensor(a!)", "dispatch": "True", "default": "False"}
```

Cuda kernel参考实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/MaxMinElementwiseKernel.cu#L14

如果是Nan值，直接返回，通过a!=a来判断是否Nan值。

Dtypes（不需要支持complex类型，其他都支持）

1. Bool类型
2. AT_DISPATCH_INTEGRAL_TYPES 所有整数类型
3. AT_DISPATCH_FLOATING_TYPES_AND2 所有浮点数类型

```
代码块
1  int32, int8, uint8, int16, int64, float32, bfloat16, half, double, bool
```

mean，mean.dim，mean.out

参考信息

官方参考文档

- https://pytorch.org/docs/stable/generated/torch.mean.html

cuda kernel 参考实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/ReduceMomentKernel.cu#L44
- 同样是使用通用的 gpu_reduce_kernel 实现
- float16 和 bfloat16 使用 float 作为中间计算类型

- dtype cast:
  - cast1：除 self 为 float16/bfloat16 且 out 为 float 情况下，其他情况都会将 self 转换为 out dtype
  - cast2:
    - self 为 float16/bfloat16 时，在 reduce 过程中会将其中一个操作数 static_cast 为 float，out dtype 可以为 float 或者 float16/bfloat16，后者则需要 project 时转换为 float16/bfloat16
    - 因此接口定义的时候 out 的 dtype 是可能和 self 的 dtype 不一样的。

代码块

```
Tensor mean(const Tensor & self, OptionalIntArrayRef dim, bool keepdim, ::std::optional<ScalarType> dtype); // {"schema": "aten::mean.dim(Tensor self, int[1]? dim, bool keepdim=False, *, ScalarType? dtype=None) -> Tensor", "dispatch": "True", "default": "True"}
Tensor & mean_out(const Tensor & self, OptionalIntArrayRef dim, bool keepdim, ::std::optional<ScalarType> dtype, Tensor & out); // {"schema": "aten::mean.out(Tensor self, int[1]? dim, bool keepdim=False, *, ScalarType? dtype=None, Tensor(a!) out) -> Tensor(a!)", "dispatch": "True", "default": "False"}
```

Function 接口

接口定义

- 暂时保证 self 和 out 的 dtype 是一样的。后续需支持 self 为 bfloat16/float16，out 为 float 的情况。

代码块

```
void fn(
    SiDeviceHandle dev,          // Device handle
    void *out,                   // Output indices (int64_t)
    const int64_t* out_sizes,    // Output sizes [ndims]
    const int64_t* out_strides,  // Output strides [ndims]
    const void *self,            // Input data pointer
    const int64_t* self_sizes,   // Input sizes [ndims]
    const int64_t* self_strides, // Input strides [ndims]
    int ndims,                   // Number of dimensions
    int* dim,                    // Reduction dimensions
    int dim_size,                // Number of reduction dimensions
    bool keepdim = false         // Keep reduced dimension
);
```

代码块

```
a = torch.randn(1, 3)
a
tensor([[ 0.2294, -0.5481,  1.3288]])
torch.mean(a)
tensor(0.3367)

a = torch.randn(4, 4)
a
tensor([[-0.3841,  0.6320,  0.4254, -0.7384],
        [-0.9644,  1.0131, -0.6549, -1.4279],
        [-0.2951, -1.3350, -0.7694,  0.5600],
        [ 1.0842, -0.9580,  0.3623,  0.2343]])
torch.mean(a, 1)
tensor([-0.0163, -0.5085, -0.4599,  0.1807])
torch.mean(a, 1, True)
tensor([[-0.0163],
        [-0.5085],
        [-0.4599],
        [ 0.1807]])
```

ISSUE

Dtypes

- AT_ALL_TYPES: `uint8_t, int8_t, int16_t, int, int64_t, float, double`
- Half, BFloat16: `float16, bfloat16`

代码块

```
AT_DISPATCH_ALL_TYPES_AND_COMPLEX
kHalf
kBFloat16
```

初版说明

数据类型：float，bfloat16

函数接口：如果需要使用Tile Core，要求dim1_size按照1024B对齐，pad value=0

代码块

```
/// @brief  Returns the mean value of each row of the tensor in the given dimension.
/// @tparam T Datatype support [float32, bfloat16], the dataype of input and output should be the same, can be different in the future.
/// @param d_out Output tensor with shape
/// @param d_in  Input tensor with shape [dim0_size, dim1_size]
```

```
/// @param dim0_size Dimension 0 size of input tensor
/// @param dim1_size Dimension 1 size of input tensor, should be 1024B aligned for tile core calculation, pad value 0
/// @param original_dim1_size Original dimension 1 size of input tensor before pad
/// @param dim The dimension or dimensions to reduce
template <typename T>
void mean_dim(void* d_out, void* d_in, int dim0_size, int dim1_size, int origianl_dim1_size, int dim);
```

sub.Tensor,sub.out,sub_.Tensor

实际直接调用的是 add_stub，无需 kernel 单独实现，依赖 add.Tensor

代码块

```
TORCH_IMPL_FUNC(sub_out) (
  const Tensor& self, const Tensor& other, const Scalar& alpha, const Tensor& result
) {
  add_stub(device_type(), *this, -alpha);
  TORCH_INTERNAL_ASSERT(result.scalar_type() == output().dtype());
}
```

代码块

```
// schema: sub.out(Tensor self, Tensor other, *, Scalar alpha=1, Tensor(a!) out) -> Tensor(a!)

// schema: sub.Tensor(Tensor self, Tensor other, *, Scalar alpha=1) -> Tensor

// schema: sub_.Tensor(Tensor(a!) self, Tensor other, *, Scalar alpha=1) -> Tensor(a!)
```

pow_.Scalar,pow.Tensor_Scalar,pow.Tensor_Scalar_out

Pytorch 文档：

- https://pytorch.org/docs/stable/generated/torch.pow.html
- 这三个 api 仅用于 exponent 为 scalar 的情况。

Pytorch 实现：

- meta

    - self 与 exponent 均为 IntegralType 且 exponent < 0 时，会直接报错
    - 获取 result_type，将 self 转换为 result_type
- Impl
    - https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/PowKernel.cu#L146
    - 对 exp 为 0.5， -0.5， -1， 2， 3， -2 时做特殊处理，其他直接调用 gpu_kernel + pow functor

接口定义

- 可统一实现一个也兼容 pow_。Tensor，pow.Tensor_Tensor，pow.Tensor_Tensor_out 的 kernel

代码块

```
Tensor & pow_out(const Tensor & self, const Scalar & exponent, Tensor & out); // {"schema": "aten::pow.Tensor_Scalar_out(Tensor self, Scalar exponent, *, Tensor(a!) out) -> Tensor(a!)", "dispatch": "True", "default": "False"}
Tensor pow(const Tensor & self, const Scalar & exponent); // {"schema": "aten::pow.Tensor_Scalar(Tensor self, Scalar exponent) -> Tensor", "dispatch": "True", "default": "True"}
Tensor & pow_(Tensor & self, const Scalar & exponent); // {"schema": "aten::pow_.Scalar(Tensor(a!) self, Scalar exponent) -> Tensor(a!)", "dispatch": "True", "default": "True"}
```

Dtypes

- AT_ALL_TYPES: `uint8_t, int8_t, int16_t, int, int64_t, float, double`
- Half, BFloat16: `float16, bfloat16`
- isComplexType: `c10::complex<c10::Half>, c10::complex<float>, c10::complex<double>`

代码块

```
AT_DISPATCH_ALL_TYPES_AND2(kHalf, kBFloat16
isComplexType
```

pow_.Tensor,pow.Tensor_Tensor,pow.Tensor_Tensor_out,pow.Scalar,pow.Scalar_out

Pytorch 文档：

- https://pytorch.org/docs/stable/generated/torch.pow.html
- 对应 exponent 为 tensor 的情况

Pytorch 实现：

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/PowKernel.cu#L99
- 支持 self 或者 exponent 为 cpu scalar，两者不同时为 cpu scalar
- 可考虑跟 pow_。Scalar 的 kernel 实现统一，类似 add.Tensor 传 bool is_exponent_scalar，bool is_self_scalar 来兼容两种形式
- 其中 pow.Scalar， pow.Scalar_out 实际上调用到 pow.Tensor_Tensor_out 的实现

代码块

```
1  Tensor & pow_out(const Tensor & self, const Tensor & exponent, Tensor & out);
   // {"schema": "aten::pow.Tensor_Tensor_out(Tensor self, Tensor exponent, *,
   Tensor(a!) out) -> Tensor(a!)", "dispatch": "True", "default": "False"}
2  Tensor pow(const Tensor & self, const Tensor & exponent); // {"schema":
   "aten::pow.Tensor_Tensor(Tensor self, Tensor exponent) -> Tensor", "dispatch":
   "True", "default": "True"}
3  Tensor & pow_(Tensor & self, const Tensor & exponent); // {"schema":
   "aten::pow_.Tensor(Tensor(a!) self, Tensor exponent) -> Tensor(a!)",
   "dispatch": "True", "default": "True"}
4  Tensor & pow_out(const Scalar & self, const Tensor & exponent, Tensor & out);
   // {"schema": "aten::pow.Scalar_out(Scalar self, Tensor exponent, *,
   Tensor(a!) out) -> Tensor(a!)", "dispatch": "True", "default": "False"}
5  Tensor pow(const Scalar & self, const Tensor & exponent); // {"schema":
   "aten::pow.Scalar(Scalar self, Tensor exponent) -> Tensor", "dispatch":
   "True", "default": "True"}
```

Dtypes

- AT_ALL_TYPES: `uint8_t, int8_t, int16_t, int, int64_t, float, double`
- AT_COMPLEX_TYPES: `c10::complex<float>, c10::complex<double>`
- kHalf, kBFloat16

代码块

```
1  AT_DISPATCH_ALL_TYPES_AND_COMPLEX_AND2(
2          kHalf, kBFloat16
```

triu,triu_,triu.out,tril,tril_,tril.out

Pytorch 文档：

- https://pytorch.org/docs/stable/generated/torch.triu.html
- https://pytorch.org/docs/stable/generated/torch.tril.html

Pytorch 实现：

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/TriangularOps.cu#L155
- triu / tril 共用同个 kernel，通过 bool upper 做标识

代码块

```
Tensor & triu_out(const Tensor & self, int64_t diagonal, Tensor & out); // {"schema": "aten::triu.out(Tensor self, int diagonal=0, *, Tensor(a!) out) -> Tensor(a!)", "dispatch": "True", "default": "False"}
Tensor triu(const Tensor & self, int64_t diagonal); // {"schema": "aten::triu(Tensor self, int diagonal=0) -> Tensor", "dispatch": "True", "default": "True"}
Tensor & triu_(Tensor & self, int64_t diagonal); // {"schema": "aten::triu_(Tensor(a!) self, int diagonal=0) -> Tensor(a!)", "dispatch": "True", "default": "True"}
Tensor & tril_out(const Tensor & self, int64_t diagonal, Tensor & out); // {"schema": "aten::tril.out(Tensor self, int diagonal=0, *, Tensor(a!) out) -> Tensor(a!)", "dispatch": "True", "default": "False"}
Tensor tril(const Tensor & self, int64_t diagonal); // {"schema": "aten::tril(Tensor self, int diagonal=0) -> Tensor", "dispatch": "True", "default": "True"}
Tensor & tril_(Tensor & self, int64_t diagonal); // {"schema": "aten::tril_(Tensor(a!) self, int diagonal=0) -> Tensor(a!)", "dispatch": "True", "default": "True"}
```

注意 dim >= 3 的 case

代码块

```
>>> a = torch.randn((2, 3, 5))
>>> a
tensor([[[ 1.1182, -0.1174,  0.5041, -0.6076, -1.5491],
         [ 0.2768,  0.1626, -0.2481, -1.2557,  1.3977],
         [ 0.0719,  0.3547,  0.5944, -0.4056,  1.6179]],

        [[ 1.1097,  0.2569, -1.0501, -0.8111, -1.1674],
         [-0.8123,  0.5775, -2.4633,  0.4311, -0.3557],
         [ 0.4450, -0.0715, -2.1173,  1.5523, -0.7991]]])
>>> torch.triu(a, -1)
tensor([[[ 1.1182, -0.1174,  0.5041, -0.6076, -1.5491],
```

```
          [ 0.2768,  0.1626, -0.2481, -1.2557,  1.3977],
          [ 0.0000,  0.3547,  0.5944, -0.4056,  1.6179]],

         [[ 1.1097,  0.2569, -1.0501, -0.8111, -1.1674],
          [-0.8123,  0.5775, -2.4633,  0.4311, -0.3557],
          [ 0.0000, -0.0715, -2.1173,  1.5523, -0.7991]]])
```

Dtypes

- AT_ALL_TYPES: `uint8_t, int8_t, int16_t, int, int64_t, float, double`
- AT_COMPLEX_TYPES: `c10::complex<float>, c10::complex<double>`
- Half, Bool, BFloat16, ComplexHalf: `float16, bool, bfloat16, c10::complex<c10::Half>`

代码块

```
AT_DISPATCH_ALL_TYPES_AND_COMPLEX_AND4(
      at::ScalarType::ComplexHalf,
      at::ScalarType::Half,
      at::ScalarType::BFloat16,
      at::ScalarType::Bool
```

masked_fill_.Scalar,masked_fill.Scalar,masked_fill_.Tensor,masked_fill.Tensor

Pytorch 文档

- https://pytorch.org/docs/stable/generated/torch.Tensor.masked_fill_.html#torch.Tensor.masked_fill_
- https://pytorch.ac.cn/docs/stable/generated/torch.Tensor.masked_fill_.html

Pytorch 实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/Indexing.cu#L1549
- 如要求 contiguous 以及 aligned，torch_sipu 会通过 TensorIteratorBridge 来保证 self 和 mask 为 contiguous 且 aligned 且 shape 一致
- value 为 Tensor 时会直接拿到 value 的 scalar 值
- 直接调用 gpu_kernel 实现

代码块

```
1  Tensor & masked_fill_(Tensor & self, const Tensor & mask, const Scalar &
   value); // {"schema": "aten::masked_fill_.Scalar(Tensor(a!) self, Tensor mask,
   Scalar value) -> Tensor(a!)", "dispatch": "True", "default": "False"}
2  Tensor masked_fill(const Tensor & self, const Tensor & mask, const Scalar &
   value); // {"schema": "aten::masked_fill.Scalar(Tensor self, Tensor mask,
   Scalar value) -> Tensor", "dispatch": "True", "default": "True"}
3  Tensor & masked_fill_(Tensor & self, const Tensor & mask, const Tensor &
   value); // {"schema": "aten::masked_fill_.Tensor(Tensor(a!) self, Tensor mask,
   Tensor value) -> Tensor(a!)", "dispatch": "True", "default": "False"}
4  Tensor masked_fill(const Tensor & self, const Tensor & mask, const Tensor &
   value); // {"schema": "aten::masked_fill.Tensor(Tensor self, Tensor mask,
   Tensor value) -> Tensor", "dispatch": "True", "default": "True"}
5  Tensor & masked_fill_out(const Tensor & self, const Tensor & mask, const
   Scalar & value, Tensor & out); // {"schema":
   "aten::masked_fill.Scalar_out(Tensor self, Tensor mask, Scalar value, *,
   Tensor(a!) out) -> Tensor(a!)", "dispatch": "True", "default": "True"}
6  Tensor & masked_fill_out(const Tensor & self, const Tensor & mask, const
   Tensor & value, Tensor & out); // {"schema":
   "aten::masked_fill.Tensor_out(Tensor self, Tensor mask, Tensor value, *,
   Tensor(a!) out) -> Tensor(a!)", "dispatch": "True", "default": "True"}
```

Dtypes

- AT_ALL_TYPES: `uint8_t, int8_t, int16_t, int, int64_t, float, double`
- AT_COMPLEX_TYPES: `c10::complex<float>, c10::complex<double>`
- Half, Bool, BFloat16, ComplexHalf: `float16, bool, bfloat16, c10::complex<c10::Half>`

代码块

```
1  AT_DISPATCH_ALL_TYPES_AND_COMPLEX_AND4(
2        kBool, kHalf, kBFloat16, kComplexHalf
```

_local_scalar_dense

无需 kernel 实现，依赖 runtime api cudaMemcpyAsync，cudaStreamSynchronize

Pytorch 实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/CUDAScalar.cu#L45
- https://github.com/pytorch/pytorch/blob/v2.5.0/c10/cuda/CUDAFunctions.h#L77

代码块

```
# 间接调用
>>> x = torch.tensor([42])
>>> x.item()
 [call] op=[aten::item], key=[PythonTLSSnapshot]
  [redispatchBoxed] op=[aten::item], key=[AutogradCPU]
   [call] op=[aten::_local_scalar_dense], key=[PythonTLSSnapshot]
    [redispatchBoxed] op=[aten::_local_scalar_dense], key=[AutogradCPU]
     [redispatch] op=[aten::_local_scalar_dense], key=[Python]
      [callBoxed] op=[aten::_local_scalar_dense], key=[Python]
       [callBoxed] op=[aten::_local_scalar_dense], key=[CPU]
42
```

cos,cos_,cos.out

Pytorch 官方文档参考：https://pytorch.org/docs/stable/generated/torch.cos.html

参考实现：

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/UnaryGeometricCosKernel.cu

代码块

```
// schema: cos(Tensor self) -> Tensor
// schema: cos_(Tensor(a!) self) -> Tensor(a!)
// schema: cos.out(Tensor self, *, Tensor(a!) out) -> Tensor(a!)

/**
 * @brief Computes the cosine of each element in the input tensor `input` and stores the result in the output tensor `out`.
 *
 * @param dev            Device on which the operation is performed.
 * @param out            Pointer to the output tensor where the cosine values will be stored.
 * @param input          Pointer to the input tensor containing values to compute cosine.
 * @param dims           Number of dimensions in the input/output tensor.
 * @param sizes          Sizes of each dimension of the input/output tensor.
 * @param strides_out    Strides of each dimension of the output tensor.
 * @param strides_input  Strides of each dimension of the input tensor.
 * @param element_sizes  Optional pointer to the element sizes of each tensor (if nullptr, strides are in bytes).
 */
void fn(dev, const void* out, const void* input, int dims, const int64_t* sizes, const int64_t* strides_out, const int64_t* strides_input, const int64_t* element_sizes=nullptr)
//updated
```

```
19  void fn(dev, const void* out, const void* self, int ndims, const int64_t*
    sizes, const int64_t* out_strides, const int64_t* self_strides, const int64_t*
    element_sizes=nullptr)
```

- [ ] Dtype 支持
  - Floating types + complex types: `float, double, at::Half, at::BFloat16, c10::complex<float>, c10::complex<double>, c10::complex<c10::Half>`

sin,sin_,sin.out

Pytorch 官方文档参考：https://pytorch.org/docs/stable/generated/torch.sin.html

参考实现：

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/UnaryGeometricSinKernel.cu

Function 接口，Dtype 支持同 cos

arange.start_out, arange, arange.out, arange.start, arange.start_step

参考信息

文档：

https://pytorch.org/docs/stable/generated/torch.arange.html

参考实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/RangeFactories.cu#L213

代码块

```
1  参考用法
2  # 创建一个预分配的输出张量
3  out = torch.empty(5, dtype=torch.float32)
4
5  # 使用 arange.start_out 填充输出张量
6  torch.arange(0, 5, out=out)
7  print(out)  # tensor([0., 1., 2., 3., 4.])
8
9  # 重用同一个输出张量
10 torch.arange(5, 10, out=out)
11 print(out)  # tensor([5., 6., 7., 8., 9.])
12
13 #指定步长
14 out = torch.empty(5, dtype=torch.float32)
15 torch.arange(0, 10, step=2, out=out)
```

```
print(out)  # tensor([0., 2., 4., 6., 8.])
```

c++

```
// build/aten/src/ATen/RegisterCompositeExplicitAutograd.cpp

m.impl("arange",
TORCH_FN(wrapper_CompositeExplicitAutograd__arange));
m.impl("arange.out",
TORCH_FN(wrapper_CompositeExplicitAutograd_out_arange_out));
m.impl("arange.start",
TORCH_FN(wrapper_CompositeExplicitAutograd_start_arange));
m.impl("arange.start_step",
TORCH_FN(wrapper_CompositeExplicitAutograd_start_step_arange));

namespace {
at::Tensor wrapper_CompositeExplicitAutograd__arange(const at::Scalar & end,
::std::optional<at::ScalarType> dtype, ::std::optional<at::Layout> layout,
::std::optional<at::Device> device, ::std::optional<bool> pin_memory) {
    // No device check
  // DeviceGuard omitted
  return at::native::arange(end, dtype, layout, device, pin_memory);
}
} // anonymous namespace
namespace {
at::Tensor & wrapper_CompositeExplicitAutograd_out_arange_out(const at::Scalar
& end, at::Tensor & out) {
    // No device check
  // DeviceGuard omitted
  return at::native::arange_out(end, out);
}
} // anonymous namespace
namespace {
at::Tensor wrapper_CompositeExplicitAutograd_start_arange(const at::Scalar &
start, const at::Scalar & end, ::std::optional<at::ScalarType> dtype,
::std::optional<at::Layout> layout, ::std::optional<at::Device> device,
::std::optional<bool> pin_memory) {
    // No device check
  // DeviceGuard omitted
  return at::native::arange(start, end, dtype, layout, device, pin_memory);
}
} // anonymous namespace
namespace {
at::Tensor wrapper_CompositeExplicitAutograd_start_step_arange(const
at::Scalar & start, const at::Scalar & end, const at::Scalar & step,
```

```
::std::optional<at::ScalarType> dtype, ::std::optional<at::Layout> layout,
::std::optional<at::Device> device, ::std::optional<bool> pin_memory) {
    // No device check
  // DeviceGuard omitted
  return at::native::arange(start, end, step, dtype, layout, device,
pin_memory);
}
} // anonymous namespace

//aten/src/ATen/native/TensorFactories.cpp
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ arange
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Tensor arange(const Scalar& end,
    std::optional<ScalarType> dtype,
    std::optional<Layout> layout,
    std::optional<Device> device,
    std::optional<bool> pin_memory) {
  return native::arange(/*start=*/0, end, dtype, layout, device, pin_memory);
}

Tensor arange(const Scalar& start, const Scalar& end,
    std::optional<ScalarType> dtype,
    std::optional<Layout> layout,
    std::optional<Device> device,
    std::optional<bool> pin_memory) {
  return native::arange(
      start, end, /*step=*/1, dtype, layout, device, pin_memory);
}

Tensor arange(
    const Scalar& start,
    const Scalar& end,
    const Scalar& step,
    std::optional<ScalarType> dtype,
    std::optional<Layout> layout,
    std::optional<Device> device,
    std::optional<bool> pin_memory) {
  // See [Note: hacky wrapper removal for TensorOptions]
  TensorOptions options =
TensorOptions().dtype(dtype).layout(layout).device(device).pinned_memory(pin_me
mory);

  bool set_to_integral_dtype = !options.has_dtype() &&
      // bool inputs are considered integral
      start.isIntegral(true) &&
      end.isIntegral(true) &&
```

```
        step.isIntegral(true);

  Tensor result = set_to_integral_dtype
      ? at::empty({0}, options.dtype(at::ScalarType::Long))
      : at::empty({0}, options);
  return at::arange_out(result, start, end, step);
}

Tensor& arange_out(const Scalar& end, Tensor& result) {
  return at::arange_out(result, /*start=*/0, end, /*step=*/1);
}


最终都dispatch到at::arange_out cuda的实现在上面链接中
```

Function 接口

代码块

```
// schema: arange.start_out(Scalar start, Scalar end, Scalar step=1, *, Tensor(a!) out) -> Tensor(a!)
template<typename T, typename T_acc_type>
/**
 * @brief Creates a 1-D tensor of size ceil((end - start) / step) with values from start to end with step step
 *
 * @param dev           Device handle for CUDA/CPU device management
 * @param start         Starting value of the sequence
 * @param end           Upper limit of the sequence (exclusive)
 * @param step          Spacing between values (must be nonzero)
 * @param dtype         Data type of output tensor
 * @param out           Optional pre-allocated output tensor (can be non-contiguous)
 *
 * @note Memory Layout:
 *       - Fully supports non-contiguous output tensors
 *       - For non-contiguous tensors, computation is done in a temporary contiguous tensor
 *         and then copied to the output tensor
 *       - This approach ensures optimal performance while supporting any memory layout
 *
 * @return Tensor       The output tensor
 */
template<typename T>
```

```
void fn_arange(
    SiDeviceHandle *dev,
    void* out,
    const T& start,
    const T& end,
    const T& step
);
支持数据类型:
整数类型
at::ScalarType::Byte (uint8)
at::ScalarType::Char (int8)
at::ScalarType::Short (int16)
at::ScalarType::Int (int32)
at::ScalarType::Long (int64)
浮点类型
at::ScalarType::Float (float32)
at::ScalarType::Double (float64)
低精度类型
at::ScalarType::Half (float16)
at::ScalarType::BFloat16 (bfloat16)
```

- [ ] 非 contiguous 的是否能支持
  - [ ] cuda 实现：检查输出张量是否连续，如果不是，它会创建一个临时的连续张量，填充数据，然后将结果复制回原始非连续张量。
  - [ ] 可以在传入 kernel 前进行判断，如果是 contiguous，可直接传入 kernel，否则在 pytorch 分配一块连续张量传入 kernel，然后将填充后的 tensor copy 给原始非 contiguous tensor。

argmax,argmax.out

参考信息

Pytorch 官方文档参考：
https://pytorch.org/docs/stable/generated/torch.argmax.html#torch.argmax

参考实现：
https://github.com/pytorch/pytorch/blob/main/aten/src/ATen/native/cuda/ReduceArgMaxKernel.cu

https://github.com/pytorch/pytorch/blob/7bd2e3bca1809a586a2b3072a16c75e871a256b3/aten/src/ATen/native/cuda/Reduce.cuh#L1173

```
代码块
```

```python
import torch

# 1. 基本用法 - 1D张量
x = torch.tensor([1, 2, 3, 4, 2])
indices = torch.argmax(x)
print(indices)  # tensor(3) - 最大值4的索引位置

# 2. 指定维度 - 2D张量
x = torch.tensor([[1, 2, 3],
                  [3, 1, 2]])
# 按行找最大值(dim=1)
indices = torch.argmax(x, dim=1)
print(indices)  # tensor([2, 0]) - 每行最大值的索引

# 3. 保持维度(keepdim=True)
indices = torch.argmax(x, dim=1, keepdim=True)
print(indices)  # tensor([[2],
                #         [0]])

# 4. 处理相同的最大值(返回第一个)
x = torch.tensor([1, 4, 4, 3])
indices = torch.argmax(x)
print(indices)  # tensor(1) - 返回第一个4的位置

# 5. 在GPU上使用
if torch.cuda.is_available():
    x_cuda = x.cuda()
    indices = torch.argmax(x_cuda)
    print(indices.device)  # cuda:0

# 6. 处理不同数据类型
x = torch.tensor([1.1, 2.2, 3.3], dtype=torch.float16)
indices = torch.argmax(x)
print(indices)  # tensor(2)

# 7. 多维张量
x = torch.randn(2, 3, 4)
# 在最后一维上找最大值
indices = torch.argmax(x, dim=-1)
print(indices.shape)  # torch.Size([2, 3])

# 8. 全局最大值
x = torch.tensor([[1, 2], [3, 4]])
index = torch.argmax(x)  # 展平后找最大值
print(index)  # tensor(3) - 4的位置

# 9. 带掩码的argmax
```

```
mask = torch.tensor([True, False, True, True])
x = torch.tensor([1, 2, 3, 4])
indices = torch.argmax(x * mask.float())
print(indices)  # tensor(3)
```

Function 接口

代码块

```
// schema: argmax(Tensor self, int? dim=None, bool keepdim=False) -> Tensor
// schema: argmax.out(Tensor self, int? dim=None, bool keepdim=False, *, Tensor(a!) out) -> Tensor(a!)
//at::Tensor & wrapper_CUDA_argmax_out_out(const at::Tensor & self, ::std::optional<int64_t> dim, bool keepdim, at::Tensor & out)

/**
 * @brief Find indices of maximum values along a dimension
 *
 * @param dev           Device handle for CUDA device management
 * @param out           Output indices tensor (int64_t)
 * @param out_sizes     Output tensor sizes [ndims]
 * @param out_strides   Output tensor strides [ndims]
 * @param self          Input tensor data pointer
 * @param self_sizes    Input tensor sizes [ndims]
 * @param self_strides  Input tensor strides [ndims]
 * @param ndim          Number of dimensions
 * @param dim           Dimension to reduce along (-1 means flatten)
 * @param dtype         Data type of input tensor
 * @param keepdim       Whether to keep reduced dimension
 *
 * @note Memory Layout:
 *       - Output tensor contains indices of maximum values
 *       - For float16/bfloat16 inputs, computation is done in float32
 *       - Indices are 64-bit integers
 *
 * @note Performance Optimizations:
 *       - Uses vectorized loads when possible
 *       - Special handling for contiguous tensors
 *       - Different implementations for inner/outer dimensions
 *
 * @note Constraints:
 *       - dim must be valid (-ndim <= dim < ndim)
 *       - Memory must be properly aligned
 *       - Maximum dimensions: 32
 *
 * @return int 0 on success, error code otherwise
 */
```

```
void fn(
    SiDeviceHandle dev,          // Device handle
    void *out,                   // Output indices (int64_t)
    const int64_t* out_sizes,    // Output sizes [ndims]
    const int64_t* out_strides,  // Output strides [ndims]
    const void *self,            // Input data pointer
    const int64_t* self_sizes,   // Input sizes [ndims]
    const int64_t* self_strides, // Input strides [ndims]
    int ndims,                    // Number of dimensions
    int dim,                     // Reduction dimension
    int dtype,                   // Input data type
    bool keepdim = false         // Keep reduced dimension
);

//支持 Half 和 BFloat16 类型，但是使用 float 作为中间计算类型
//其余基本数据类型均使用本身的数据类型作为中间计算类型
- uint8_t    // Byte
- int8_t     // Char
- int16_t    // Short
- int32_t    // Int
- int64_t    // Long
- float      // Float
- double     // Double
```

ISSUE

- [ ] Dtype 支持 //cuda 的实现 支持 *Byte Char Short Int Long Float Double*
  - [ ] float32
  - [x] ~~float16 使用 float 作为中间计算类型~~
  - [x] ~~bfloat16 使用 float 作为中间计算类型~~
  - [ ] mxformat
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
  - [ ] 需要额外存储 stride 和 size 信息，用来进行复杂的索引计算
- [ ] Memory 是否需要 Align，Align 到多少

isin.Tensor_Tensor,isin.Tensor_Tensor_out

参考信息

https://pytorch.org/docs/stable/generated/torch.isin

```
In [188]: (c,d,torch.isin(c, d))
 [call] op=[aten::isin.Tensor_Tensor], key=[AutogradCUDA]
  [redispatch] op=[aten::isin.Tensor_Tensor], key=[CUDA]
   [call] op=[aten::fill_.Scalar], key=[CUDA]
   [call] op=[aten::unsqueeze], key=[CUDA]
    [call] op=[aten::as_strided], key=[CUDA]
   [call] op=[aten::view], key=[CUDA]
   [call] op=[aten::eq.Tensor], key=[CUDA]
   [call] op=[aten::any.dim], key=[CUDA]
    [call] op=[aten::as_strided], key=[CUDA]
   [call] op=[aten::copy_], key=[CUDA]
```

```
(tensor([[    1, 65536],
         [    2,     3]], device='cuda:0'),
 tensor([   22, 65536], device='cuda:0'),
 tensor([[False,  True],
         [False, False]], device='cuda:0'))
```

参考实现：

https://github.com/pytorch/pytorch/blob/3f069e7679588d5ee4b1d5b2492ca0e20f9320b5/aten/src/ATen/native/cuda/TensorCompare.cpp#L11

代码块

```python
import torch

# 创建测试数据
elements = torch.tensor([1, 2, 3, 4, 5])
test_elements = torch.tensor([2, 4, 6])

# 检查 elements 中的元素是否在 test_elements 中
result = torch.isin(elements, test_elements)
print(result)  # 输出: tensor([False, True, False, True, False])

# 使用 Tensor 方法
result = elements.isin(test_elements)
print(result)  # 输出: tensor([False, True, False, True, False])

# 使用 assume_unique 参数
result = torch.isin(elements, test_elements, assume_unique=True)

# 使用 invert 参数获取不在 test_elements 中的元素掩码
result = torch.isin(elements, test_elements, invert=True)
print(result)  # 输出: tensor([True, False, True, False, True])
```

Function 接口

代码块

```
- func: isin.Tensor_Tensor_out(Tensor elements, Tensor test_elements, *, bool assume_unique=False, bool invert=False, Tensor(a!) out) -> Tensor(a!)
  variants: function
```

```
  structured: True
  dispatch:
    CPU, CUDA: isin_Tensor_Tensor_out
    MPS: isin_Tensor_Tensor_out_mps

 //isin.Tensor_Tensor_out(Tensor elements, Tensor test_elements, *, bool assume_unique=False, bool invert=False, Tensor(a!) out) -> Tensor(a!)


  /**
 * @brief Test each element in 'elements' for membership in 'test_elements'
 *
 * @param dev           Device handle for device management
 * @param elements      Input tensor to check
 * @param test_elements Elements to test for membership
 * @param assume_unique If True, assumes both elements and test_elements contain unique elements
 * @param invert        If True, return nonzero where element is NOT in test_elements
 * @param out           Output boolean tensor (pre-allocated)
 * @param dtype         Data type of input tensors(optional)
 *
 * @note Functionality:
 *       - Returns boolean tensor of same shape as elements
 *       - True where element is in test_elements, False otherwise
 *       - If invert=True, returns opposite
 *
 * @note Performance:
 *       - For GPU: Materializes cross product for parallel comparison
 *       - assume_unique=True can enable optimizations
 *       - Uses tensor broadcasting internally
 *
 * @note Example:
 *       elements = [1, 2, 3, 4]
 *       test_elements = [2, 4, 6]
 *       out = [False, True, False, True]
 *
 * @return int 0 on success, error code otherwise
 */
void fn(
    SiDeviceHandle dev,          // Device handle
    const void *elements,        // Input elements to check
    const void *test_elements,   // Test elements to compare against
    bool assume_unique,          // Whether inputs contain unique elements
    bool invert,                 // Whether to invert the result
    void *out,                    // Output boolean tensor
```

```
    int Type(optional)          // Data type of input tensors (elements & test_elements)
);
```

ISSUE

- [ ] Dtype 支持 //cuda 的实现 支持 kComplexHalf， kHalf， kBool， kBFloat16
  - [ ] float32
  - [ ] float16
  - [ ] bfloat16
  - [ ] mxformat
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
  - [ ] 非连续在具体实现中也是会被转换为 contiguous，不会涉及到 stride，dim 信息
- [ ] Memory 是否需要 Align，Align 到多少

cat.out, cat

参考信息

Pytorch 官方文档参考：https://pytorch.org/docs/stable/generated/torch.cat.html#torch.cat

并行优化实现参考：
https://github.com/pytorch/pytorch/blob/32f585d9346e316e554c8d9bf7548af9f62141fc/aten/src/ATen/native/cuda/Shape.cu#L288C6-L288C18

串行实现版本参考：
https://github.com/pytorch/pytorch/blob/32f585d9346e316e554c8d9bf7548af9f62141fc/aten/src/ATen/native/cuda/Shape.cu#L526

代码块

```
int64_t offset = 0; // 跟踪当前复制位置
for (const Tensor& t : materialized) {
  if (cat_should_skip_tensor(t)) continue;
  // 获取当前张量在连接维度上的大小
  int64_t dimSize = t.size(dim);
  // 从结果张量中切出对应大小的视图
  Tensor nt = at::narrow(result, dim, offset, dimSize);
  // 将当前张量复制到视图中
  copy_(nt, t);
  //更新偏移量
  offset += dimSize;
}

```

```
//example:
// 例如连接两个张量 [2,3] 和 [2,4] 在维度1上:
tensor1 = [[1,2,3],       result = [[1,2,3,4,5,6,7],
           [4,5,6]]                 [8,9,10,11,12,13,14]]
tensor2 = [[4,5,6,7],
           [11,12,13,14]]

// 步骤:
1. 创建结果张量 result，大小为 [2,7]
2. narrow 操作切出前3列: nt1 = result[:, 0:3]
3. 复制第一个张量: copy_(nt1, tensor1)
4. narrow 操作切出后4列: nt2 = result[:, 3:7]
5. 复制第二个张量: copy_(nt2, tensor2)
```

Function 接口

代码块

```
//schema :
//cat.out(Tensor[] tensors, int dim=0, *, Tensor(a!) out) -> Tensor(a!)
//cat(Tensor[] tensors, int dim=0) -> Tensor
# alias for torch.cat
//concat(Tensor[] tensors, int dim=0) -> Tensor
//concat.out(Tensor[] tensors, int dim=0, *, Tensor(a!) out) -> Tensor(a!)
//concat.names(Tensor[] tensors, Dimname dim) -> Tensor
//concat.names_out(Tensor[] tensors, Dimname dim, *, Tensor(a!) out) ->
Tensor(a!)
# alias for torch.cat
//concatenate(Tensor[] tensors, int dim=0) -> Tensor
//concatenate.out(Tensor[] tensors, int dim=0, *, Tensor(a!) out) -> Tensor(a!)
//concatenate.names(Tensor[] tensors, Dimname dim) -> Tensor
//concatenate.names_out(Tensor[] tensors, Dimname dim, *, Tensor(a!) out) ->
Tensor(a!)

//概括起来拥有两种调用方式
//使用整数作为维度参数
"concat(TensorList tensors, int64_t dim=0, *, Tensor out=None)"
//使用命名维度(Dimname)作为维度参数
"concat(TensorList tensors, Dimname dim, *, Tensor out=None)"

// 原始的实现，对于每个函数，有四种情况:
case 0: {
  if (_r.isNone(2)) {
    // 1. 使用整数维度，无输出张量
    return at::concat(tensors, dim);
  } else {
    // 2. 使用整数维度，有输出张量
```

```
      return at::concat_out(out, tensors, dim);
    }
  }
  case 1: {
    if (_r.isNone(2)) {
      // 3. 使用命名维度，无输出张量
      return at::concat(tensors, dim);
    } else {
      // 4. 使用命名维度，有输出张量
      return at::concat_out(out, tensors, dim);
    }
  }
```

代码块

```
//接口中不包含Dimname，需要提前处理转换为dim，默认输出存放在out指针，不考虑是否return output
/**
 * @brief Concatenate multiple tensors along the specified dimension
 *
 * @param dev              Device handle
 * @param out              Output tensor data pointer (pre-allocated)
 * @param out_ndims        Number of outs' dimensions
 * @param out_sizes        Array of output tensor sizes for each dimension
 * @param out_strides      Array of output tensor strides for each dimension
 * @param tensors          Array of input data pointers
 * @param num_tensors      Number of input tensors
 * @param tensors_sizes    Array of input sizes [num_tensors * tensors_ndims]
 * @param tensors_strides  Array of input strides [num_tensors * tensors_ndims]
 * @param dim              Concatenation dimension
 * @param element_size     element_size of tensor
 * @param tensors_ndims    Number of temsors' dimensions (<= 4)
 * @param all_contiguous   Whether all inputs are memory contiguous
 * @param all_same_dtype   Whether all inputs have same data type
 * @param all_same_stride  Whether all inputs have same stride pattern
 * @param memory_format    Memory format (0: Contiguous, 1: ChannelsLast, 2: ChannelsLast3d) (optional)
 *
 * @note Memory Layout:
 *       - tensors_size[i][j]->[(i-1)*tensors_ndim+j]: size of i-th tensor in j-th dimension
 *       - tensors_stride[i][j]->[(i-1)*tensors_ndim+j]: stride of i-th tensor in j-th dimension
 *
 * @note Performance Optimizations:
 *       - Fast path when all_contiguous is true
```

```
 *       - Special handling for aligned memory
 *       - Batch processing when all_same_dtype is true
 *
 * @note Constraints:
 *       - Maximum dimensions: 4
 *       - All tensors must be 32-bit indexable
 *
 * @return int 0 on success, error code otherwise
 */
void fn(
    SiDeviceHandle dev,             // Device handle
    void *out,                      // Output data pointer
    const int64_t out_ndims,         // Outputs' dimensions
    const int64_t* out_sizes,        // Output sizes [out_ndim]
    const int64_t* out_strides,      // Output strides [out_ndim]
    void **tensors,                 // Array of input data pointers [num_tensors]
    int num_tensors,                // Number of input tensors
    const int64_t tensors_ndims,     // tensorimensions
    const int64_t* tensors_sizes,    // Input sizes [num_tensors* tensors_ndim]
    const int64_t* tensors_strides,  // Input strides [num_tensors* tensors_ndim]
    int dim,                        // Concatenation dimension
    int element_size,                // element_size of tensor
    bool all_contiguous,            // Whether all inputs are contiguous
    bool all_same_dtype,            // Whether all inputs have same dtype
    bool all_same_stride,           // Whether all inputs have same stride
    int memory_format               // Memory format (optional)
);

// input_sizes, input_strides说明，假设连接两个 2x3 张量
int64_t out_size[] = {2, 6};  // 2x6 输出
int64_t out_stride[] = {6, 1};  // 连续存储

int64_t tensors_size[] = {
    2, 3,  // 第一个输入
    2, 3   // 第二个输入
};
int64_t tensors_stride[] = {
    3, 1,  // 第一个输入
    3, 1   // 第二个输入
};

//cuda 的实现 支持 kComplexHalf, kHalf, kBool, kBFloat16
```

ISSUE

- [ ] Dtype 支持 //cuda 的实现 支持 kComplexHalf， kHalf， kBool， kBFloat16
  - [ ] float32
  - [ ] float16
  - [ ] bfloat16
  - [ ] mxformat
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
  - [ ] 非连续情况使用较小的批处理大小（CAT_ARRAY_BATCH_SIZE/2）
  - [ ] 需要额外存储 stride 和 size 信息，用来进行复杂的索引计算
  - [ ] 内存格式必须是 Contiguous
- [ ] Memory 是否需要 Align，Align 到多少
  - [ ] 参考 cuda 实现中有 16 字节对齐的检查，如果对齐可以进行加速

cumsum.out, cumsum, cumsum_

参考信息

Pytorch 官方文档参考：https://pytorch.org/docs/stable/generated/torch.cumsum.html

cuda 实现参考：
https://github.com/pytorch/pytorch/blob/main/aten/src/ATen/native/cuda/CumsumKernel.cu

https://github.com/pytorch/pytorch/blob/bb7e8fbd668c7c8931436b4a935b26911cbe0daf/aten/src/ATen/native/cuda/ScanUtils.cuh#L446

代码块

```
import torch

# 1. 基本用法 - 1D张量
x = torch.tensor([1, 2, 3, 4])
y = torch.cumsum(x, dim=0)
print(y)  # tensor([1, 3, 6, 10])

# 2. 2D张量 - 按行累加
x = torch.tensor([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])
y = torch.cumsum(x, dim=0)
print(y)
# tensor([[ 1,  2,  3],
#         [ 5,  7,  9],
#         [12, 15, 18]])

```

```python
# 3. 2D张量 - 按列累加
y = torch.cumsum(x, dim=1)
print(y)
# tensor([[ 1,  3,  6],
#         [ 4,  9, 15],
#         [ 7, 15, 24]])

# 4. 指定输出类型
y = torch.cumsum(x, dim=0, dtype=torch.float)
print(y.dtype)  # torch.float32

# 5. 使用方法形式
y = x.cumsum(dim=0)

# 6. 在GPU上使用
if torch.cuda.is_available():
    x_cuda = x.cuda()
    y_cuda = torch.cumsum(x_cuda, dim=0)
    print(y_cuda.device)  # cuda:0

# 7. 处理大维度张量
x = torch.randn(2, 3, 4, 5)
y = torch.cumsum(x, dim=2)  # 在第3维上累加
print(y.shape)  # torch.Size([2, 3, 4, 5])

# 8. 负维度索引
y = torch.cumsum(x, dim=-1)  # 等同于 dim=3
```

Function 接口

代码块

```
cumsum.out(Tensor self, int dim, *, ScalarType? dtype=None, Tensor(a!) out) -> Tensor(a!)

void launch_cumsum_cuda_kernel(const TensorBase& result, const TensorBase& self, int64_t dim) {
  AT_DISPATCH_ALL_TYPES_AND_COMPLEX_AND2(
      ScalarType::Half, ScalarType::BFloat16,
      self.scalar_type(), "cumsum_cuda",
      [&]() {
        scalar_t init = 0;
        scan_dim<scalar_t>(
            self,
            result,
            dim,
            init,
```

```
            std::plus<scalar_t>());
        });
}

/**
 * @brief Compute the cumulative sum along a dimension
 *
 * @param dev           Device handle for CUDA device management
 * @param out           Output tensor data pointer (pre-allocated)
 * @param out_sizes     Output tensor sizes for each dimension [ndim]
 * @param out_strides   Output tensor strides for each dimension [ndim]
 * @param self          Input tensor data pointer
 * @param self_sizes    Input tensor sizes for each dimension [ndim]
 * @param self_strides  Input tensor strides for each dimension [ndim]
 * @param ndims         Number of dimensions
 * @param dim           Dimension to compute cumsum along
 * @param dtype         Data type of tensors
 * @param all_contiguous Whether all tensors are memory contiguous
 *
 * @note Memory Layout:
 *       - Input and output tensors must have same shape
 *       - For contiguous tensors, memory is assumed to be packed
 *       - For non-contiguous tensors, strides specify memory layout
 *
 * @note Performance Optimizations:
 *       - Special fast path when input is contiguous
 *       - Uses vectorized operations for aligned memory
 *       - Different implementations for innermost vs outer dimensions
 *
 * @note Constraints:
 *       - dim must be valid (0 <= dim < ndim)
 *       - Memory must be properly aligned (16 bytes)
 *       - Maximum dimensions: 32
 *
 */
void cumsum_fn(
    SiDeviceHandle dev,          // Device handle
    const void *out,             // Output data pointer
    const int64_t* out_sizes,    // Output sizes [ndims]
    const int64_t* out_strides,  // Output strides [ndims]
    const void *self,            // Input data pointer
    const int64_t* self_sizes,   // Input sizes [ndims]
    const int64_t* self_strides, // Input strides [ndims]
    int ndims,                   // Number of dimensions
    int dim,                     // Cumsum dimension
    int dtype,                   // Data type
    bool all_contiguous = true   // Whether all tensors are contiguous
```

```
);

support dtype:
- uint8_t    // Byte
- int8_t     // Char
- int16_t    // Short
- int32_t    // Int
- int64_t    // Long
- float      // Float
- double     // Double
- complex<float>   // ComplexFloat
- complex<double>  // ComplexDouble
- ScalarType::Half      // float16
- ScalarType::BFloat16  // bfloat16
```

ISSUE

- [ ] Dtype 支持
  - [x] ~~float32~~
  - [x] ~~float16~~
  - [x] ~~bfloat16~~
  - [ ] mxformat
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
  - [ ] For contiguous tensors, memory is assumed to be packed
  - [ ] For non-contiguous tensors, strides specify memory layout
- [ ] Memory 是否需要 Align，Align 到多少

clamp

Clamp 和 clamp_tensor 在实现逻辑上是一致的，这里设计一个统一接口。clamp_min，clamp_max 是 clamp 实现的特例，也放在一起进行处理

clamp, clamp_, clamp.out, clamp_max, clamp_max_, clamp_max.out, clamp_min, clamp_min_, clamp_min.out

clamp.Tensor, clamp_.Tensor, clamp.Tensor_out, clamp_max.Tensor, clamp_max_.Tensor, clamp_max.Tensor_out, clamp_min.Tensor, clamp_min_.Tensor, clamp_min.Tensor_out

参考信息

pytorch 官方资料：https://pytorch.org/docs/stable/generated/torch.clamp.html#torch.clamp

cuda 实现：
https://github.com/pytorch/pytorch/blob/7ae0ce6360b6e4f944906502d20da24c04debee5/aten/src/ATen/native/cuda/TensorCompare.cu#L61C13-L61C32（scalar）

https://github.com/pytorch/pytorch/blob/7ae0ce6360b6e4f944906502d20da24c04debee5/aten/src/ATen/native/cuda/TensorCompare.cu#L44(tensor)

代码块

```
// 1. 基本用法 - clamp(input, min, max)
{
    auto input = torch::tensor({-2, 0, 2, 4, 6});

    // 使用标量限制范围
    auto output = torch::clamp(input, /*min=*/0, /*max=*/4);
    // output: [0, 0, 2, 4, 4]

    // 使用 Tensor 限制范围
    auto min = torch::tensor({-1, 1, 0, 2, -2});
    auto max = torch::tensor({1, 3, 2, 4, 6});
    output = torch::clamp(input, min, max);
    // 每个元素独立限制在对应的范围内
}

// 2. 原地操作 - clamp_(input, min, max)
{
    auto input = torch::tensor({-2, 0, 2, 4, 6});

    // 原地修改
    input.clamp_(/*min=*/0, /*max=*/4);
    // input: [0, 0, 2, 4, 4]
}

// 3. 使用 out 参数 - clamp.out(input, min, max, out)
{
    auto input = torch::tensor({-2, 0, 2, 4, 6});
    auto out = torch::empty_like(input);

    torch::clamp_out(out, input, /*min=*/0, /*max=*/4);
    // out: [0, 0, 2, 4, 4]
}

// 4. 只限制最小值 - clamp_min
{
    auto input = torch::tensor({-2, 0, 2, 4, 6});

    auto output = torch::clamp_min(input, /*min=*/2);
```

```
    // output: [2, 2, 2, 4, 6]

    // 原地版本
    input.clamp_min_(2);
}

// 5. 只限制最大值 - clamp_max
{
    auto input = torch::tensor({-2, 0, 2, 4, 6});

    auto output = torch::clamp_max(input, /*max=*/2);
    // output: [-2, 0, 2, 2, 2]

    // 原地版本
    input.clamp_max_(2);
}

// 6. 处理 NaN
{
    auto input = torch::tensor({-2, NAN, 2, 4, 6});

    // NaN 会被保留
    auto output = torch::clamp(input, /*min=*/0, /*max=*/4);
    // output: [0, NaN, 2, 4, 4]
}

// 7. 不同数据类型
{
    // 支持所有数值类型
    auto input_float = torch::tensor({-2.5, 0.5, 2.5}, torch::kFloat32);
    auto output_float = torch::clamp(input_float, 0.0, 2.0);

    auto input_int = torch::tensor({-2, 0, 2}, torch::kInt32);
    auto output_int = torch::clamp(input_int, 0, 2);
}

```

Function 接口

代码块

```
//clamp.out(Tensor self, Scalar? min=None, Scalar? max=None, *, Tensor(a!) out) -> Tensor(a!)
//clamp.Tensor_out(Tensor self, Tensor? min=None, Tensor? max=None, *, Tensor(a!) out) -> Tensor(a!)

/**
```

```
 5    * @brief Clamps all elements in input into the range [min, max]
 6    *
 7    * @param dev              Device handle for device management
 8    * @param out              Output data pointer
 9    * @param self             Input data pointer
10    * @param min              Lower-bound tensor/scalar value
11    * @param max              Upper-bound tensor/scalar value
12    * @param minmax           Decide trigger clamp/clamp_min/clamp_max
13    * @param is_min_scalar    Decide the min is scalar/tensor
14    * @param is_max_scalar    Decide the max is scalar/tensor
15    *
16    * @note Supported Types:
17    *       - All floating point types (float, double)
18    *       - All integer types (uint8, int8, int16, int32, int64)
19    *       - Half precision (Half, BFloat16)
20    *
21    */
22   template<typename T>
23   void clamp_min_max_fn(
24       SiDeviceHandle dev,          // Device handle
25       void *out,                   // Output data pointer
26       const void *self,            // Input data pointer
27       const void *min,             // Lower bound tensor
28       const void *max,             // Upper bound tensor
29       int minmax,                  // mode (0 for clamp, 1 for clamp_min, 2 for
     clamp_max, tmep)
30       bool is_min_scalar,          // Decide the min is scalar/tensor, true-
     scalar false-tensor
31       bool is_max_scalar           // Decide the max is scalar/tensor, true-
     scalar false-tensor
32   );
33   //该接口参考了void add_bf16(SiDeviceHandle dev, void *d_in0, void *d_in1, void
     *d_out, int dim0_size, int dim1_size, sifmt::bfloat16 alpha, bool
     is_d_in0_scalar, bool is_d_in1_scalar);
34   //可以处理min/max可能为tensor或scalar的情况
```

ISSUE

- [ ] Dtype 支持
  - [x] ~~float32~~
  - [x] ~~float16~~
  - [x] ~~bfloat16~~
  - [ ] mxformat
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）

- [ ] 该算子是 pointwise 的，建议转换为 contiguous 后进行处理

- [ ] Memory 是否需要 Align，Align 到多少

neg.out, neg, neg_

参考信息

pytorch 官方资料：https://pytorch.org/docs/stable/generated/torch.neg.html

cuda 实现参考：
https://github.com/pytorch/pytorch/blob/94969d0a40e14eb36dd158d9805f48fb2f477cae/aten/src/ATen/native/cuda/UnarySignKernels.cu#L29

代码块

```
import torch

# 1. 基本用法
x = torch.tensor([1, -2, 3])
y = torch.neg(x)
print(y)  # tensor([-1,  2, -3])

# 2. 使用方法形式
x = torch.tensor([1, -2, 3])
y = x.neg()
print(y)  # tensor([-1,  2, -3])

# 3. 使用负号运算符
x = torch.tensor([1, -2, 3])
y = -x
print(y)  # tensor([-1,  2, -3])

# 4. 原地操作
x = torch.tensor([1, -2, 3])
x.neg_()  # 注意后缀下划线表示原地操作
print(x)  # tensor([-1,  2, -3])

# 5. 不同数据类型
x = torch.tensor([1.5, -2.5, 3.5], dtype=torch.float32)
y = torch.neg(x)
print(y)  # tensor([-1.5000,  2.5000, -3.5000])

# 6. 复数类型
x = torch.tensor([1+2j, -3-4j])
y = torch.neg(x)
print(y)  # tensor([-1.-2.j,  3.+4.j])

```

```python
# 7. 多维张量
x = torch.tensor([[1, -2], [3, -4]])
y = torch.neg(x)
print(y)  # tensor([[-1,  2], [-3,  4]])

# 8. 在GPU上使用
if torch.cuda.is_available():
    x_cuda = torch.tensor([1, -2, 3], device='cuda')
    y_cuda = torch.neg(x_cuda)
    print(y_cuda)  # tensor([-1,  2, -3], device='cuda:0')
```

Function 接口

代码块

```
at::Tensor & wrapper_CUDA_neg_out_out(const at::Tensor & self, at::Tensor & out)

/**
 * @brief Compute element-wise negation of a tensor
 *
 * @param dev           Device handle for CUDA device management
 * @param out           Output tensor data pointer
 * @param self          Input tensor data pointer
 * @param sizes         Tensor sizes [ndim]
 * @param strides       Tensor strides [ndim]
 * @param ndim          Number of dimensions
 * @param dtype         Data type of tensors
 *
 * @note Memory Layout:
 *       - Input and output must have same shape
 *       - Supports non-contiguous tensors
 *       - Optimized for contiguous case
 *
 * @note Performance:
 *       - Uses vectorized loads for aligned memory
 *       - JIT compilation for complex types
 *       - Different implementations for contiguous/strided
 *
 */
void fn(
    SiDeviceHandle dev,          // Device handle
    void *out,                   // Output data pointer
    const void *self,            // Input data pointer
    const int64_t* sizes,        // Tensor sizes [ndim]
    const int64_t* strides,      // Tensor strides [ndim]
    int ndim,                    // Number of dimensions
```

```
    int dtype                   // Data type
);

support Dtype
所有基本数值类型 (通过 AT_DISPATCH_ALL_TYPES):
uint8_t (Byte)
int8_t (Char)
int16_t (Short)
int32_t (Int)
int64_t (Long)
float (Float)
double (Double)
半精度浮点类型 (通过 ScalarType::Half):
at::Half (float16)
BFloat16 类型 (通过 ScalarType::BFloat16):
at::BFloat16 (bfloat16)
复数类型 (通过 AT_DISPATCH_COMPLEX_TYPES_AND):
c10::complex<float> (ComplexFloat)
c10::complex<double> (ComplexDouble)
c10::complex<Half> (ComplexHalf)
```

ISSUE

- [ ] Dtype 支持
  - [x] ~~float32~~
  - [x] ~~float16~~
  - [x] ~~bfloat16~~
  - [ ] mxformat
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
  - [ ] *Different implementations for contiguous/strided*
- [ ] Memory 是否需要 Align，Align 到多少

aten::as_strided

参考信息

官网文档：https://pytorch.org/docs/stable/generated/torch.as_strided.html#torch.as_strided

参考实现：
https://github.com/pytorch/pytorch/blob/7ae0ce6360b6e4f944906502d20da24c04debee5/c10/core/TensorImpl.cpp#L846

用法参考：

代码块

```
# 基本用法
import torch

# 创建一个连续的张量
x = torch.tensor([[1, 2, 3],
                  [4, 5, 6]])
print(x.size())            # torch.Size([2, 3])
print(x.stride())          # (3, 1) - 表示移动一行需要步进3个元素，移动一列需要步进1个元素

# 使用as_strided创建一个新视图
# 1. 转置操作
y = x.as_strided(size=(3, 2), stride=(1, 3))
print(y)
# 输出:
# tensor([[1, 4],
#         [2, 5],
#         [3, 6]])

# 2. 提取对角线元素
diag = x.as_strided(size=(2,), stride=(4,))
print(diag)  # tensor([1, 5])

# 3. 使用storage_offset参数
# 从第二个元素开始，创建一个1x2的视图
z = x.as_strided(size=(1, 2), stride=(3, 1), storage_offset=1)
print(z)  # tensor([[2, 3]])

# 4. 创建重叠视图
patches = x.as_strided(size=(2, 2, 2), stride=(3, 1, 1))
print(patches)
# 结果包含2x2大小的重叠窗口
# tensor([[[1, 2],
#          [2, 3]],
#         [[4, 5],
#          [5, 6]]])

# 5. 实现卷积的滑动窗口
input = torch.arange(16).reshape(4, 4)
# tensor([[ 0,  1,  2,  3],
#         [ 4,  5,  6,  7],
#         [ 8,  9, 10, 11],
#         [12, 13, 14, 15]])
windows = input.as_strided(size=(2, 2, 3, 3),
                           stride=(4, 1, 4, 1))
print(windows)  # 第一个3x3窗口
# tensor([[[[ 0,  1,  2],
```

```
#          [ 4,  5,  6],
#          [ 8,  9, 10]],
#         [[ 1,  2,  3],
#          [ 5,  6,  7],
#          [ 9, 10, 11]]],
#        [[[ 4,  5,  6],
#          [ 8,  9, 10],
#          [12, 13, 14]],
#         [[ 5,  6,  7],
#          [ 9, 10, 11],
#          [13, 14, 15]]]])
```

Function 接口

代码块

```
/**
 * @brief Creates a view of the input tensor with specified size, stride, and optional storage offset
 *
 * @param dev           Device handle for CUDA/CPU device management
 * @param self          Input tensor data pointer
 * @param self_sizes    Input tensor sizes [ndims]
 * @param self_strides  Input tensor strides [ndims]
 * @param self_ndims    Number of dimensions for input tensor
 * @param out           Output tensor data pointer (same storage as input)
 * @param size          Desired output sizes [out_ndims]
 * @param stride        Desired output strides [out_ndims]
 * @param out_ndims     Number of dimensions for output tensor
 * @param storage_offset Storage offset for output tensor (in elements)
 * @param dtype         Data type of tensor
 *
 * @note Memory Layout:
 *       - Same underlying storage is shared between input and output
 *       - No memory allocation occurs
 *       - Output tensor accesses input tensor's storage according to size/stride/offset
 *
 * @note Safety:
 *       - Caller is responsible for ensuring valid size/stride/offset combinations
 *       - Invalid parameters may cause out-of-bounds memory access
 *
 * @note Examples:
 *       1. Transpose a 2x3 tensor to 3x2:
 *          - Input: sizes=[2,3], strides=[3,1]
 *          - Output: sizes=[3,2], strides=[1,3]
```

```
 *
 *       2. Get diagonal elements of a square matrix:
 *          - Input: sizes=[NxN]
 *          - Output: sizes=[N], strides=[N+1]
 *
 *       3. Extract a subtensor with offset:
 *          - Input: sizes=[H,W]
 *          - Output: sizes=[h,w], storage_offset=(i*W+j)
 *
 */
void as_strided(
    SiDeviceHandle *dev,          // Device handle
    const void *self,             // Input tensor data pointer
    const int64_t* self_sizes,    // Input tensor sizes
    const int64_t* self_strides,  // Input tensor strides
    int self_ndims,               // Number of input dimensions
    void *out,                    // Output tensor data pointer (same as input)
    const int64_t* size,          // Output tensor sizes
    const int64_t* stride,        // Output tensor strides
    int out_ndims,                // Number of output dimensions
    int64_t storage_offset = 0,   // Storage offset (in elements)
    int dtype = kFloat            // Data type
);
```

ISSUE

- [ ] Dtype 支持
  - [ ] view 操作，不涉及到计算，应该支持所有的 type
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
- [ ] Memory 是否需要 Align，Align 到多少

eq.Scalar_out, eq.Scalar, eq_.Scalar, eq.Tensor_out, eq.Tensor, eq_.Tensor, nq.Scalar_out, nq.Scalar, nq_.Scalar, nq.Tensor_out, nq.Tensor, nq_.Tensor [TODO]

cuda 中 eq 和 nq 是通过同一个 kernel 实现的，逻辑上反转即可，scalar 和 tensor 可以通过 flag 来控制，实现统一算子接口

参考信息

https://pytorch.org/docs/stable/generated/torch.eq.html

https://pytorch.org/docs/stable/generated/torch.ne.html#torch.ne

实现参考
https://github.com/pytorch/pytorch/blob/7ae0ce6360b6e4f944906502d20da24c04debee5/aten/

src/ATen/native/cuda/CompareEQKernel.cu#L32

代码块

```
eq
# 1. 张量与张量比较
x = torch.tensor([1, 2, 3])
y = torch.tensor([1, 1, 3])
result = torch.eq(x, y)  # 或 x.eq(y) 或 x == y
# result: tensor([True, False, True])

# 2. 张量与标量比较
x = torch.tensor([1, 2, 3])
result = torch.eq(x, 2)  # 或 x.eq(2) 或 x == 2
# result: tensor([False, True, False])

ne
# 1. 张量与张量比较
x = torch.tensor([1, 2, 3])
y = torch.tensor([1, 1, 3])
result = torch.ne(x, y)  # 或 x.ne(y) 或 x != y
# result: tensor([False, True, False])

# 2. 张量与标量比较
x = torch.tensor([1, 2, 3])
result = torch.ne(x, 2)  # 或 x.ne(2) 或 x != 2
# result: tensor([True, False, True])
```

Function 接口

代码块

```
/**
 * @brief Performs element-wise equality/inequality comparison
 *
 * @param dev           Device handle for CUDA/CPU device management
 * @param out           Output tensor (bool type) to store the result
 * @param self          First input tensor
 * @param other         Second input tensor or scalar
 * @param out_sizes     output sizes
 * @param out_strides   output strides (can be non-contiguous)
 * @param self_sizes    First input sizes
 * @param self_strides  First input strides (can be non-contiguous)
 * @param other_sizes   Second input sizes (nullptr if scalar)
 * @param other_strides Second input strides (can be non-contiguous)
 * @param self_ndims         Number of self dimensions
 * @param other_ndims          Number of other dimensions
```

```
 * @param dtype         Data type of inputs
 * @param is_other_scalar Whether other is a scalar
 * @param op_type       0 for eq, 1 for ne
 *
 * @note Memory Layout:
 *       - Fully supports non-contiguous tensors
 *       - Handles arbitrary strides efficiently
 *       - Optimized for both contiguous and non-contiguous cases
 */
void compare_eq_ne(
    SiDeviceHandle *dev,           // Device handle
    void *out,                     // Output tensor data pointer (bool type)
    const void *self,              // First input tensor data pointer
    const void *other,             // Second input tensor/scalar data pointer
    const int64_t* out_sizes,      // output sizes
    const int64_t* sout_strides,   // output strides
    const int64_t* self_sizes,     // First input sizes
    const int64_t* self_strides,   // First input strides
    const int64_t* other_sizes,    // Second input sizes (nullptr if scalar)
    const int64_t* other_strides,  // Second input strides (nullptr if scalar)
    int self_ndims,                // Number of self dimensions
    int other_ndims,               // Number of other dimensions
    int dtype,                     // Data type of inputs (optional)
    bool is_other_scalar,          // Whether other is a scalar
    int op_type                    // 0 for eq, 1 for ne
);

支持的数据类型
基本数值类型
整型
kByte / kUInt8 (uint8)
kChar / kInt8 (int8)
kShort / kInt16 (int16)
kInt / kInt32 (int32)
kLong / kInt64 (int64)
浮点型
kFloat (float32)
kDouble (float64)
复数类型
kComplexFloat (complex64)
kComplexDouble (complex128)
kComplexHalf (complex32)
低精度类型
kHalf (float16)
kBFloat16 (brain floating point)
特殊类型
kBool (布尔)
```

```
AT_FLOAT8_TYPES (Float8类型，包括 kFloat8_e4m3fn 和 kFloat8_e5m2)
高级无符号类型
AT_BAREBONES_UNSIGNED_TYPES
kUInt16 (uint16)
kUInt32 (uint32)
kUInt64 (uint64)
```

ISSUE

- [ ] Dtype 支持
  - [ ] AT_EXPAND(AT_ALL_TYPES_AND_COMPLEX), kComplexHalf, kHalf, kBFloat16, kBool, AT_EXPAND(AT_FLOAT8_TYPES), AT_EXPAND(AT_BAREBONES_UNSIGNED_TYPES));
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
  - [ ] pytorch 中使用了 TensorIteratorBase 传递 self，other， out， 需考虑非 contiguous 实现
- [ ] 考虑到 broadcast 的情况，out 和 self 的 size/stride 存在不一致的情况，需要传入
- [ ] Memory 是否需要 Align，Align 到多少

fill_.Scalar, fill.Scalar_out, fill_.Tensor, fill.Tensor_out

参考信息

文档

https://pytorch.org/docs/stable/generated/torch.Tensor.fill_.html

实现：
https://github.com/pytorch/pytorch/blob/2e0c98ff058eb6661f991f05557a673187eead84/aten/src/ATen/native/cuda/FillKernel.cu#L22

代码块

```
fill_.Scalar(Tensor(a!) self, Scalar value) -> Tensor(a!)
fill_.Tensor(Tensor(a!) self, Tensor value) -> Tensor(a!)

fill_.Tensor虽然支持输入tensor，但是只能是标量张量，有限制value.dim() == 0，最终调用的还是标量的kernel

参考用法：
# 创建张量
x = torch.zeros(2, 3)
print(x)
# tensor([[0., 0., 0.],
#         [0., 0., 0.]])
```

```python

# 填充标量值
x.fill_(5)
print(x)
# tensor([[5., 5., 5.],
#         [5., 5., 5.]])

# 填充不同类型的值
y = torch.zeros(2, 3, dtype=torch.int)
y.fill_(3.14)  # 会被截断为整数
print(y)
# tensor([[3, 3, 3],
#         [3, 3, 3]])

# 创建张量
x = torch.zeros(2, 3)

# 创建填充后的新张量
y = torch.fill(x, 7)
print(x)  # 原张量不变
# tensor([[0., 0., 0.],
#         [0., 0., 0.]])
print(y)  # 新张量被填充
# tensor([[7., 7., 7.],
#         [7., 7., 7.]])

# 创建标量张量
scalar_tensor = torch.tensor(9.5)

# 使用标量张量填充
x = torch.zeros(2, 3)
x.fill_(scalar_tensor)
print(x)
# tensor([[9.5000, 9.5000, 9.5000],
#         [9.5000, 9.5000, 9.5000]])
```

Function 接口

代码块

```
/**
 * @brief Fills a tensor with a scalar value
 *
 * @param dev           Device handle for CUDA/CPU device management
 * @param self          Tensor data pointer to be filled
 * @param self_sizes    Tensor sizes [ndims]
```

```
 * @param self_strides  Tensor strides [ndims] (supports non-contiguous tensors)
 * @param ndims         Number of dimensions
 * @param value         Scalar value to fill the tensor with
 * @param dtype         Data type of the tensor (optional)
 *
 * @note Supported Types:
 *       - All numeric types (float, int, etc.)
 *       - Complex types
 *       - Boolean type
 *       - Half precision (float16, bfloat16)
 *       - Float8 types
 *       - Unsigned types
 *
 * @note Memory Layout:
 *       - Supports both contiguous and non-contiguous tensors
 *       - Handles arbitrary strides efficiently
 *
 * @note Special Cases:
 *       - For single-element CPU tensors, a specialized fast path is used
 *
 * @return int 0 on success, error code otherwise
 */
int fill(
    SiDeviceHandle *dev,          // Device handle
    void *self,                   // Tensor data pointer
    const int64_t* self_sizes,    // Tensor sizes
    const int64_t* self_strides,  // Tensor strides
    int ndims,                    // Number of dimensions
    const void* value,            // Scalar value pointer
    int dtype                     // Data type of tensor (optional)
);
```

ISSUE

- [ ] Dtype 支持
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
  - [ ] pytorch 中使用了 TensorIteratorBase 传递 self， 需考虑非 contiguous 实现
- [ ] Memory 是否需要 Align，Align 到多少
- [ ] 对于 tensor/scalar
  - [ ] 如果使用 fill_stub，会通过 pytorch 的原始实现来对 tensor 进行处理，可以直接获取（TensorIterator& *iter*， const Scalar& *value*）作为参数

- [ ] 如果不使用 fill_stub，需要自己手动处理不同的参数，对 tensor 进行判断，提取标量后和 scalar 进行相同处理

div.out, div_.Tensor, div.Tensor, div.Scalar, div_.Scalar

参考信息

参考实现：
https://github.com/pytorch/pytorch/blob/9cd52da45c3030601bdd27e8a948f23be12b21ee/aten/src/ATen/native/cuda/BinaryDivTrueKernel.cu#L20

资料：https://pytorch.org/docs/stable/generated/torch.div.html#torch.div

cuda 中只注册了 div.out， div_。Tensor， div.Tensor，cuda 通过 div_true_stub 注册 div，cuda 的 kernel 中有针对第二个输入是 scalar 的情况优化，会将 scalar 取倒数后按照乘法计算

代码块

```
# 张量与张量除法
x = torch.tensor([10.0, 20.0, 30.0])
y = torch.tensor([2.0, 5.0, 10.0])

# 三种等价方式
result1 = torch.div(x, y)
result2 = x / y
result3 = x.div(y)

print(result1)  # tensor([5.0000, 4.0000, 3.0000])

a = torch.tensor([10.0, 20.0, 30.0])
c = torch.div(a, 2.0)
print(c)  # tensor([5.0000, 10.0000, 15.0000])

# 使用运算符
c = a / 2.0
print(c)  # tensor([5.0000, 10.0000, 15.0000])
```

Function 接口

代码块

```
div.Tensor(Tensor self, Tensor other) -> Tensor
div_.Tensor(Tensor(a!) self, Tensor other) -> Tensor(a!)
div.out(Tensor self, Tensor other, *, Tensor(a!) out) -> Tensor(a!)

/**
```

```
 6   * @brief Performs element-wise division of tensor by another tensor or scalar
 7   *
 8   * @param dev           Device handle for CUDA/CPU device management
 9   * @param self          Dividend tensor (numerator)
10   * @param other         Divisor tensor or scalar (denominator)
11   * @param self_sizes    Tensor sizes of self [ndims]
12   * @param self_strides  Tensor strides of self [ndims]
13   * @param other_sizes   Tensor sizes of other [ndims] (nullptr if scalar)
14   * @param other_strides Tensor strides of other [ndims] (nullptr if scalar)
15   * @param ndims         Number of dimensions
16   * @param out           Output tensor (can be same as self for in-place)
17   * @param out_sizes     Output tensor sizes [ndims]
18   * @param out_strides   Output tensor strides [ndims]
19   * @param is_self_scalar
20   *
21   * @note Supported Types:
22   *       - Floating point types (float, double)
23   *       - Complex types (complex<float>, complex<double>)
24   *       - Half precision (float16, bfloat16)
25   *
26   * @note Performance Optimizations:
27   *       - Special fast path for scalar divisors
28   *       - For scalar division, computes a * (1/b) instead of a / b
29   *       - Handles non-contiguous tensors efficiently
30   *
31   * @note Broadcast Support:
32   *       - Automatically handles broadcasting between tensors
33   *       - Caller must ensure output tensor has correct broadcast shape
34   *
35   * @return int 0 on success, error code otherwise
36   */
37  int div(
38      SiDeviceHandle dev,
39      const void* self,
40      const void* other,
41      const int64_t* self_sizes,
42      const int64_t* self_strides,
43      const int64_t* other_sizes,
44      const int64_t* other_strides,
45      int self_ndims,
46      int other_ndims,
47      void* out,
48      const int64_t* out_sizes,
49      const int64_t* out_strides
50      bool is_self_scalar
51  );
52
```

```
支持数据类型
浮点类型：
float32 (torch.float)
float64 (torch.double)
float16 (torch.half)
bfloat16 (torch.bfloat16)
复数类型：
complex64 (torch.complex64)
complex128 (torch.complex128)
CUDA 实现不直接支持整数类型（如 int8, int16, int32, int64），因为在 PyTorch 中，整数除法会返回浮点结果（真除法行为）。
```

首个版本开发说明

（1）数制支持上，初版需要支持float，bf16类型

（2）除数（输入2）需要支持tensor与scalar两种形式

（3）输出与被除数（输入1）的shape一致

（4）如果除数为tensor，那么除数与被除数shape应相同（如不同，则由框架处理进行broadcast），做elementwise除法

（5）对齐限制：硬件目前要求dim0_size * sizeof(T)为1024B对齐

（6）输入输出参数，均由框架进行预处理（reshape为一维并padding至符合对齐限制）

内部接口调用形式：

代码块

```
template <typename T>
void div(void *d_in0, void *d_in1, void *d_out, int dim0_size, bool is_d_in1_scalar);
```

ISSUE

- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
  - [ ] 需要考虑到 broadcast 和第二个输入为 scalar 的情况。需要 stride，size，dim 信息

exponential_

参考信息

参考实现：
https://github.com/pytorch/pytorch/blob/main/aten/src/ATen/native/cuda/DistributionExponentialKernel.cu

https://github.com/pytorch/pytorch/blob/6055a4f612782ca944f2e0465f7497b7f18de4e9/aten/src/ATen/native/cuda/DistributionTemplates.h#L558

资料：https://pytorch.org/docs/stable/generated/torch.Tensor.exponential_.html

代码块

```python
import torch

# 创建一个张量
x = torch.tensor([1.0, 2.0, 3.0])

# 原地进行指数变换
x.exponential_()

print(x)  # 输出: tensor([ 2.7183,  7.3891, 20.0855])
```

Function 接口

代码块

```
(Tensor(a!) self, float lambd=1, *, Generator? generator=None) -> Tensor(a!)

/**
 * @brief Fills the input tensor with samples from an exponential distribution.
 *
 * @param self          The output tensor to fill with samples.
 * @param lambda        The rate parameter (lambda) of the exponential distribution.
 * @param gen           Optional random number generator for reproducibility.
 *
 * @return void*        ptr to the output tensor filled with samples.
 */

 void* fn(
    SiDeviceHandle *dev,
    void* self,
    double lambda,
    std::optional<Generator> gen = std::nullopt
    const int64_t* self_sizes,
    const int64_t* self_strides,
    int ndims,
);

 (update)


```

```
支持的数据类型：
浮点类型：float32, float64, float16, bfloat16
复数类型：complex64, complex128
```

ISSUE

- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
  - [ ] 通过 exponential_stub 注册，支持非 contiguous 数据，需要提供 size，stride，dim 信息

addmm.out,addmm,mm.out,mm, _addmm_activation.out, _addmm_activation

参考信息

文档：https://pytorch.org/docs/stable/generated/torch.mm.html

https://pytorch.org/docs/stable/generated/torch.addmm.html

Addmm:

$$\text{out} = \beta\ \text{input} + \alpha\ (\text{mat1}_i \mathbin{@} \text{mat2}_i)$$

参考实现：

https://github.com/pytorch/pytorch/blob/919d54b7b1bcba4dfd425675f7c29438e1435469/aten/src/ATen/native/cuda/Blas.cpp#L318

代码块

```
TORCH_IMPL_FUNC(addmm_out_cuda)(const Tensor& self, const Tensor& mat1, const Tensor& mat2, const Scalar& beta, const Scalar& alpha, const Tensor& result) {
  addmm_out_cuda_impl(const_cast<Tensor&>(result), self, mat1, mat2, beta, alpha);
}

TORCH_IMPL_FUNC(addmm_activation_out_cuda)(const Tensor& self, const Tensor& mat1, const Tensor& mat2, const Scalar& beta, const Scalar& alpha, bool use_gelu, const Tensor& result) {
  addmm_out_cuda_impl(const_cast<Tensor&>(result), self, mat1, mat2, beta, alpha, use_gelu ? Activation::GELU : Activation::RELU);
}

TORCH_IMPL_FUNC(mm_out_cuda)(const Tensor& self, const Tensor& mat2, const Tensor& result) {
  addmm_out_cuda_impl(const_cast<Tensor&>(result), result, self, mat2, 0, 1);
```

```
}

三个op通过同一个kernel实现

>>> A = torch.tensor([[1, 2], [3, 4]])
>>> B = torch.tensor([[5, 6], [7, 8]])
>>> input = torch.tensor([[1, 1], [1, 1]])

>>> C = torch.mm(A,B)
>>> C
tensor([[19, 22],
        [43, 50]])

>>> C = torch.addmm(input, A, B)
>>> C
tensor([[20, 23],
        [44, 51]])

>>> C = torch.addmm(input, A, B, beta=1, alpha=2.0)
>>> C
tensor([[ 39,  45],
        [ 87, 101]])

torch._addmm_activation(input, mat1, mat2, beta=0.5, alpha=2.0,use_gelu=True)
等价于
gelu(beta * input + alpha * torch.mm(mat1, mat2))

torch._addmm_activation(input, mat1, mat2, beta=0.5, alpha=2.0,use_gelu=False)
等价于
relu(beta * input + alpha * torch.mm(mat1, mat2))

>>> result = torch._addmm_activation(input, mat1, mat2, use_gelu=True)
>>> equivalent = gelu(input + torch.mm(mat1, mat2))
>>> result
tensor([[ 2.7273e+00,  1.2938e+00,  4.7260e-02, -2.6163e-03],
        [ 1.1547e+00, -1.3293e-01, -9.3054e-02,  1.9921e-01],
        [-1.1569e-04,  2.0479e+00,  6.1474e-01,  1.1607e+00]])
>>> equivalent
tensor([[ 2.7273e+00,  1.2938e+00,  4.7260e-02, -2.6163e-03],
        [ 1.1547e+00, -1.3293e-01, -9.3054e-02,  1.9921e-01],
        [-1.1569e-04,  2.0479e+00,  6.1474e-01,  1.1607e+00]])

>>> result = torch._addmm_activation(input, mat1, mat2, beta=0.5,
alpha=2.0,use_gelu=True)
>>> equivalent = gelu(0.5 * input + 2.0 * torch.mm(mat1, mat2))
>>> result
tensor([[ 3.1910e+00, -1.6997e-01, -6.1870e-02, -1.6937e-02],
```

```
        [ 2.7823e+00, -2.6890e-02, -3.9319e-03, -1.6966e-01],
        [-6.1636e-06,  4.7033e+00,  2.5606e+00,  3.8821e+00]])
>>> equivalent
tensor([[ 3.1910e+00, -1.6997e-01, -6.1870e-02, -1.6937e-02],
        [ 2.7823e+00, -2.6890e-02, -3.9319e-03, -1.6966e-01],
        [-6.1636e-06,  4.7033e+00,  2.5606e+00,  3.8821e+00]])
```

Function 接口

代码块

```
/**
 * @brief 执行矩阵乘法并加偏置，可选激活函数: out = beta * bias + alpha * (A @ B)
 *
 * @param dev           设备句柄，用于 CUDA 设备管理
 * @param a_ptr         第一个矩阵的数据指针 (A)
 * @param b_ptr         第二个矩阵的数据指针 (B)
 * @param bias_ptr      偏置张量的数据指针，用于加到矩阵乘法结果
 * @param a_sizes       A 矩阵的维度 [2]，形状为 (m, k)
 * @param a_strides     A 矩阵的步长 [2]
 * @param b_sizes       B 矩阵的维度 [2]，形状为 (k, n)
 * @param b_strides     B 矩阵的步长 [2]
 * @param bias_sizes    偏置张量的维度，可以是 [2] 或 [1]
 * @param bias_strides  偏置张量的步长
 * @param bias_ndims    偏置张量的维数 (1 或 2)
 * @param out_ptr       输出张量的数据指针
 * @param out_sizes     输出张量的维度 [2]，形状为 (m, n)
 * @param out_strides   输出张量的步长 [2]
 * @param dtype         所有张量的数据类型 ID
 * @param alpha         A @ B 的缩放因子
 * @param beta          偏置的缩放因子
 * @param activation    激活函数类型: 0=None, 1=ReLU, 2=GELU
 *
 * @note 支持的数据类型:
 *       - 浮点类型 (float, double)
 *       - 半精度类型 (half, bfloat16)
 *       - 8位浮点类型 (float8_e4m3fn, float8_e5m2)
 *
 * @note 内存布局:
 *       - 支持各种内存布局 (行优先、列优先、非连续)
 *       - 自动处理转置和共轭转置
 *       - 如果 A 或 B 是非连续的，可能会产生临时副本
 *
 * @note 张量广播:
 *       - 偏置可以是向量 (1D) 或矩阵 (2D)
 *       - 如果偏置是向量，会被广播到结果形状
 *
```

```
 */
void addmm_fn(
    SiDeviceHandle dev,
    const void* a_ptr,
    const void* b_ptr,
    const void* bias_ptr,
    const int64_t* a_sizes,
    const int64_t* a_strides,
    const int64_t* b_sizes,
    const int64_t* b_strides,
    const int64_t* bias_sizes,
    const int64_t* bias_strides,
    int bias_ndims,
    void* out_ptr,
    const int64_t* out_sizes,
    const int64_t* out_strides,
    int dtype,
    float alpha,
    float beta,
    int activation
);

默认a,b,out的ndims均为2

支持的输入类型
1. 浮点类型
float32 (torch.float, at::kFloat) - 单精度浮点
float64 (torch.double, at::kDouble) - 双精度浮点
2. 半精度类型
float16 (torch.half, at::kHalf) - 半精度浮点
bfloat16 (torch.bfloat16, at::kBFloat16) - Brain 浮点格式
3. 低精度类型（特定 GPU 架构支持）
float8_e4m3fn (torch.float8_e4m3fn, at::kFloat8_e4m3fnuz) - 8位浮点 (4位指数，3位尾数)
float8_e5m2 (torch.float8_e5m2, at::kFloat8_e5m2fnuz) - 8位浮点 (5位指数，2位尾数)
```

ISSUE

- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
  - [ ] addmm_out_cuda_impl 使用 prepare_matrix_for_cublas 方法来处理非连续输入

index.Tensor,index.Tensor_out, index_put_impl, _index_put_impl, _index_put_impl.out, index_put_, index_put.out

参考信息

文档：

https://pytorch.org/docs/stable/generated/torch.Tensor.index_put_.html#torch.Tensor.index_put_

参考实现：

https://github.com/pytorch/pytorch/blob/981807cfcbc2dc8036f0e83f038455b49ba4b833/aten/src/ATen/native/cuda/IndexKernel.cu#L196

https://github.com/pytorch/pytorch/blob/981807cfcbc2dc8036f0e83f038455b49ba4b833/aten/src/ATen/native/cuda/IndexKernel.cu#L248

代码块

```
>>> x = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> row_indices = torch.tensor([0, 2])  # 选择第0行和第2行
>>> result = x[row_indices]
>>> result
tensor([[1, 2, 3],
        [7, 8, 9]])
>>> row_indices = torch.tensor([0, 1, 2])
>>> col_indices = torch.tensor([1, 2, 0])
>>> result = x[row_indices, col_indices]
>>> result
tensor([2, 6, 7])
>>>

>>> x = torch.zeros(3, 3)  # 默认是 float 类型
>>> indices = (torch.tensor([0, 2]), torch.tensor([1, 1]))
>>> values = torch.tensor([10, 20], dtype=torch.float)  # 显式指定为浮点类型
>>> result = x.index_put(indices, values)
>>>
>>> result
tensor([[ 0., 10.,  0.],
        [ 0.,  0.,  0.],
        [ 0., 20.,  0.]])

>>> x = torch.ones(3, 3)  # 浮点类型
>>> x
tensor([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]])
>>> indices = (torch.tensor([0, 0]), torch.tensor([1, 1]))  # 索引位置 (0,1) 重复
>>> values = torch.tensor([10.0, 20.0])  # 浮点类型匹配 x
>>> result = x.index_put(indices, values, accumulate=True)
>>> result
tensor([[ 1., 31.,  1.],
```

```
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.]])
>>>

cuda实现中共用一个入口
template <typename scalar_t>
void index_kernel_impl(TensorIteratorBase& iter, const IntArrayRef index_size, const IntArrayRef index_stride) {
  gpu_index_kernel(iter, index_size, index_stride, []C10_DEVICE(char* const out_data, const char* const in_data, const int64_t offset) {
    *reinterpret_cast<scalar_t*>(out_data) = *reinterpret_cast<const scalar_t*>(in_data + offset);
  });
}

template <typename scalar_t>
void index_put_kernel_impl(TensorIterator& iter, const IntArrayRef index_size, const IntArrayRef index_stride) {
  gpu_index_kernel(iter, index_size, index_stride, []C10_DEVICE(char* const out_data, const char* const in_data, const int64_t offset) {
    *reinterpret_cast<scalar_t*>(out_data + offset) = *reinterpret_cast<const scalar_t*>(in_data);
  });
}
```

Function 接口

代码块

```
/**
 * @brief 执行高级索引操作 (index 和 index_put)
 *
 * @param dev             设备句柄，管理 CUDA 上下文
 * @param out             结果张量 (对于 index 是输出，对于 index_put 是被索引的目标)
 * @param indices         索引张量数组，每个维度一个索引，使用 nullptr 表示不索引的维度
 * @param values
 * @param self            源张量 (对于 index 是被索引的源，对于 index_put 是输入值)
 * @param out_ndims
 * @param out_sizes
 * @param out_strides
 * @param indices_ndims
 * @param indices_sizes   每个索引张量的尺寸数组，表示有效索引范围
 * @param indices_strides 每个索引张量的步长数组，配合索引使用
 * @param num_indices     索引张量的数量
```

```
 * @param values_ndims
 * @param values_sizes
 * @param values_strides
 * @param self_ndims
 * @param self_sizes
 * @param self_strides
 * @param is_put           true=index_put 操作，false=index 操作
 * @param accumulate       当 is_put=true 时是否累加值而不是替换（仅对 index_put 有效）
 *
 * @note 支持的索引类型：
 *       - 整数索引张量：选择特定索引位置的数据
 *       - nullptr：表示该维度不进行索引，相当于整个维度都被选择
 *       - 布尔掩码：通过布尔值为 true 的位置进行索引
 *
 * @note 边界处理：
 *       - 索引从 0 到 (dimension_size-1)
 *       - 负索引在内部被修正：-1 表示最后一个元素
 *       - 索引超出范围时会抛出异常
 *
 * @note 内存布局：
 *       - 支持非连续输入和输出张量
 *       - 内部高效处理各种跨步和偏移计算
 *
 * @return int 0表示成功，非零表示错误码
 */
void fn(
    SiDeviceHandle dev,
    const void* self,
    const void** indices,
    const void* values,
    void* out,
    const int64_t self_ndims,
    const int64_t* self_sizes,
    const int64_t* self_strides,
    const int64_t* indices_ndims,
    const int64_t* indices_sizes,
    const int64_t* indices_strides,
    int64_t num_indices,
    const int64_t values_ndims,
    const int64_t* values_sizes,
    const int64_t* values_strides,
    const int64_t out_ndims,
    const int64_t* out_sizes,
    const int64_t* out_strides,
    bool is_put,
    bool accumulate
```

```
);

index 操作支持：
所有基本类型：int8, int16, int32, int64, uint8, float, double
所有复数类型：complex64, complex128
半精度类型：float16 (kHalf)
布尔类型：bool (kBool)
BFloat16：bfloat16 (kBFloat16)
复数半精度：complex32 (kComplexHalf)

index_put 操作支持所有 index 支持的类型，并额外支持：
8位浮点类型：float8_e4m3fn, float8_e5m2 (AT_FLOAT8_TYPES)
```

ISSUE

- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
- [x] ~~CUDA 实现的 index_put 不支持 accumulate=true 参数~~

gt.Tensor_out gt.Tensor gt_.Tensor gt.Scalar_out gt.Scalar gt_.Scalar, lt.Tensor_out lt.Tensor lt_.Tensor lt.Scalar_out lt.Scalar lt_.Scalar

代码块

```
//Cuda中 gt lt le使用了同一个接口，通过设定一个flag来确认具体的method

void gt_kernel_cuda(TensorIteratorBase& iter) {
  compare_kernel_with_scalars(iter, OpType::GT);
}

void le_kernel_cuda(TensorIteratorBase& iter) {
  compare_kernel_with_scalars(iter, OpType::LE);
}

void lt_kernel_cuda(TensorIteratorBase& iter) {
  compare_kernel_with_scalars(iter, OpType::LT);
}
```

gt.Tensor_out gt.Tensor gt_.Tensor gt.Scalar_out gt.Scalar gt_.Scalar

参考信息

参考实现：

https://github.com/pytorch/pytorch/blob/9cd52da45c3030601bdd27e8a948f23be12b21ee/aten/src/ATen/native/cuda/CompareKernels.cu#L86

资料：

https://pytorch.org/docs/stable/generated/torch.gt.html

代码块

```
gt.Scalar_out(Tensor self, Scalar other, *, Tensor(a!) out) -> Tensor(a!)
gt.Tensor_out(Tensor self, Tensor other, *, Tensor(a!) out) -> Tensor(a!)

# 张量与张量比较
x = torch.tensor([1, 2, 3, 4])
y = torch.tensor([2, 2, 2, 2])

# 三种等价方式
result1 = torch.gt(x, y)
result2 = x > y
result3 = x.gt(y)

print(result1)  # tensor([False, False, True, True])


x = torch.tensor([1, 2, 3, 4])
# 比较张量中的每个元素是否大于2
result = x > 2
print(result)  # tensor([False, False, True, True])


# 不同形状的张量比较（使用广播）
x = torch.tensor([[1, 2], [3, 4]])
y = torch.tensor([2, 1])

result = x > y
print(result)
# tensor([[False, True],
#         [True, True]])
```

lt.Tensor_out lt.Tensor lt_.Tensor lt.Scalar_out lt.Scalar lt_.Scalar

参考信息

参考实现：

https://github.com/pytorch/pytorch/blob/9cd52da45c3030601bdd27e8a948f23be12b21ee/aten/src/ATen/native/cuda/CompareKernels.cu#L94

文档：https://pytorch.org/docs/stable/generated/torch.lt.html

代码块

```python
import torch

a = torch.tensor([1, 2, 3, 4])
b = torch.tensor([2, 2, 2, 2])

# 返回布尔张量 运算符形式
result = a < b
print(result)  # tensor([True, False, False, False])


# 张量与张量比较
a = torch.tensor([1, 2, 3, 4])
b = torch.tensor([2, 2, 2, 2])
result = torch.lt(a, b)
print(result)  # tensor([True, False, False, False])

# 张量与标量比较
result = torch.lt(a, 2)
print(result)  # tensor([True, False, False, False])

# 批量比较
batch = torch.tensor([[1, 2, 3], [4, 5, 6]])
threshold = torch.tensor([2, 4, 3])

result = batch < threshold
print(result)
# tensor([[ True, True, False],
#         [False, False, False]])
```

le.Scalar_out, le.Scalar, le.Tensor_out, le.Tensor

文档：https://pytorch.org/docs/stable/generated/torch.le.html

代码块

```python
import torch

# 创建两个张量
a = torch.tensor([1, 2, 3, 4])
b = torch.tensor([2, 2, 2, 2])

# 使用 torch.le 比较
result = torch.le(a, b)
print("a <= b:", result)
# a <= b: tensor([True, True, False, False])

# 使用张量方法比较
```

```python
result_method = a.le(b)
print("a.le(b):", result_method)
# a.le(b): tensor([True, True, False, False])

# 创建一个张量
x = torch.tensor([[1, 2, 3], [4, 5, 6]])

# 与标量比较
result = torch.le(x, 3)
print(result)
# tensor([[ True,  True,  True],
#         [False, False, False]])

# 不同形状的张量比较
a = torch.tensor([[1, 2], [3, 4]])
b = torch.tensor([2, 1])  # 将被广播为 [[2, 1], [2, 1]]

result = torch.le(a, b)
print(result)
# tensor([[ True, False],
#         [False, False]])
```

Function 接口

代码块

```
/**
 * @brief Performs element-wise gt/lt/le comparison
 *
 * @param dev           Device handle for CUDA/CPU device management
 * @param out           Output tensor (bool type) to store the result
 * @param self          First input tensor
 * @param other         Second input tensor or scalar
 * @param out_sizes     output sizes
 * @param out_strides   output strides (can be non-contiguous)
 * @param self_sizes    First input sizes
 * @param self_strides  First input strides (can be non-contiguous)
 * @param other_sizes   Second input sizes (nullptr if scalar)
 * @param other_strides Second input strides (can be non-contiguous)
 * @param self_ndims          Number of self dimensions
 * @param other_ndims         Number of other dimensions
 * @param dtype         Data type of inputs
 * @param is_other_scalar Whether other is a scalar
 * @param OpType        0 for gt, 1 for lt, 2 for le
 *
 * @note Memory Layout:
 *       - Fully supports non-contiguous tensors
```

```
 *       - Handles arbitrary strides efficiently
 *       - Optimized for both contiguous and non-contiguous cases
 */
void compare_gt(
    SiDeviceHandle *dev,           // Device handle
    void *out,                     // Output tensor data pointer (bool type)
    const void *self,              // First input tensor data pointer
    const void *other,             // Second input tensor/scalar data pointer
    const int64_t* out_sizes,      // output sizes
    const int64_t* sout_strides,   // output strides
    const int64_t* self_sizes,     // First input sizes
    const int64_t* self_strides,   // First input strides
    const int64_t* other_sizes,    // Second input sizes (nullptr if scalar)
    const int64_t* other_strides,  // Second input strides (nullptr if scalar)
    int self_ndims,                // Number of self dimensions
    int other_ndims,               // Number of other dimensions
    int dtype,                     // Data type of inputs (optional)
    bool is_other_scalar,          // Whether other is a scalar
    int OpType                     // 0 for gt, 1 for lt, 2 for le
);
```

ISSUE

- [ ] Dtype 支持
  - [ ] AT_EXPAND(AT_ALL_TYPES_AND_COMPLEX), kComplexHalf, kHalf, kBFloat16, kBool, AT_EXPAND(AT_FLOAT8_TYPES), AT_EXPAND(AT_BAREBONES_UNSIGNED_TYPES));
- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
  - [ ] pytorch 中使用了 TensorIteratorBase 传递 self，other， out， 需考虑非 contiguous 实现
- [ ] 考虑到 broadcast 的情况，out 和 self 的 size/stride 存在不一致的情况，需要传入
- [ ] Memory 是否需要 Align，Align 到多少

resize_, resize_output, resize.out, resize

参考信息

文档：https://pytorch.org/docs/stable/generated/torch.Tensor.resize_.html

参考实现：
https://github.com/pytorch/pytorch/blob/32f585d9346e316e554c8d9bf7548af9f62141fc/aten/src/ATen/native/cuda/Resize.cpp#L17

可以观察到在 cuda 的实现实际上就是 **cudaMemcpyAsync**

代码块

```python
import torch

# 创建张量
x = torch.randn(2, 3)
print(f"原始张量: {x.shape}")
# 原始张量: torch.Size([2, 3])

# 调整大小为 3x2
x.resize_(3, 2)
print(f"调整后张量: {x.shape}")
# 调整后张量: torch.Size([3, 2])

# 创建一个张量
x = torch.tensor([1, 2, 3, 4, 5, 6]).view(2, 3)
print(f"原始张量:\n{x}")
# 原始张量:
# tensor([[1, 2, 3],
#         [4, 5, 6]])


# 增加大小 - 新元素值不可预测!
x.resize_(3, 3)
print(f"增加大小后(新值不确定):\n{x}")
# 增加大小后(新值不确定):
# tensor([[1, 2, 3],
#         [4, 5, 6],
#         [?, ?, ?]])  # 新元素的值是未定义的


# 减少大小 - 截断元素
y = torch.tensor([1, 2, 3, 4, 5, 6]).view(2, 3)
y.resize_(1, 3)
print(f"减少大小后:\n{y}")
# 减少大小后:
# tensor([[1, 2, 3]])

tensor = torch.tensor([1, 2, 3, 4, 5, 6])

# 1. resize_: 实际改变底层存储，可增加或减少元素
tensor_resized = tensor.clone()
tensor_resized.resize_(2, 4)  # 增加元素，新元素未初始化

# 2. view: 仅改变视图，总元素数必须相同
```

```python
tensor_view = tensor.view(2, 3)  # 必须是相同元素数

# 3. reshape: 类似view，但可处理非连续张量
tensor_reshaped = tensor.reshape(3, 2)  # 可能创建副本

# 4. contiguous+view: 确保连续后改变视图
tensor_cont = tensor.contiguous().view(2, 3)

# 大多数情况下，推荐使用 view，reshape 或 permute 来调整张量形状，这些方法更安全且保留梯度关系。resize_ 主要用于需要实际改变内存分配的特殊场景。
```

cuda 的处理流程（大部分工作在 torch 侧）

1. 入口点：resize_cuda_函数

代码块

```
const Tensor& resize_cuda_(const Tensor& self, IntArrayRef size, std::optional<MemoryFormat> optional_memory_format) {
  // 处理具名张量
  if (self.has_names()) {
    return resize_named_tensor_(self, size, optional_memory_format);
  }

  // 记录旧存储大小（用于确定性算法）
  auto* self_ = self.unsafeGetTensorImpl();
  int64_t old_storage_nbytes = self_->unsafe_storage() ? self_->unsafe_storage().nbytes() : 0;

  // 调整实际大小
  resize_impl_cuda_(self_, size, /*strides=*/std::nullopt);

  // 处理内存格式调整
  if (optional_memory_format.has_value()) {
    // ...重新设置步长
    self_->empty_tensor_restride(memory_format);
  }

  // 确定性内存初始化（如果需要）
  if (C10_UNLIKELY(at::globalContext().deterministicAlgorithms() && ...)) {
    at::native::fill_resize_deterministic_(self, old_storage_nbytes);
  }

  return self;
}
```

2. 核心实现：resize_impl_cuda_函数

代码块

```
inline TensorImpl* resize_impl_cuda_(TensorImpl* self, IntArrayRef size, at::OptionalIntArrayRef stride) {
  // 如果大小和步长没变，直接返回
  if (self->sizes() == size && (!stride || self->strides() == stride)) {
    return self;
  }

  // 计算元素大小和存储偏移
  const auto itemsize = self->dtype().itemsize();
  const auto storage_offset = self->storage_offset();

  size_t storage_size = 1;
  if (stride) {
    // 使用给定步长设置大小
    self->set_sizes_and_strides(size, *stride);
    // 计算需要的存储字节数
    storage_size = at::detail::computeStorageNbytes(size, *stride, itemsize, storage_offset);
  } else {
    // 设置连续内存布局
    self->set_sizes_contiguous(size);
    // 计算连续内存所需字节数
    storage_size = at::detail::computeStorageNbytesContiguous(size, itemsize, storage_offset);
  }

  // 根据需要调整存储大小
  maybe_resize_storage_cuda(self, storage_size);

  return self;
}
```

3. 存储扩容检查：maybe_resize_storage_cuda 函数

代码块

```
static inline void maybe_resize_storage_cuda(TensorImpl* self, size_t new_size_bytes) {
  // 空张量不需要调整存储
  if (self->numel() == 0) {
    return;
  }

```

```
  const Storage &storage = self->unsafe_storage();
  TORCH_CHECK(storage, "Tensor: invalid null storage");

  // 仅当需要更大存储时才调整
  if (new_size_bytes > storage.nbytes()) {
    resize_bytes_cuda(storage.unsafeGetStorageImpl(), new_size_bytes);
  }
}
```

4. 实际内存调整：resize_bytes_cuda 函数

代码块

```
void resize_bytes_cuda(StorageImpl* storage, size_t size_bytes) {
  // 检查存储是否可调整大小
  TORCH_CHECK(storage->resizable(), "Trying to resize storage that is not resizable");
  auto allocator = storage->allocator();
  TORCH_CHECK(allocator != nullptr, "Trying to resize storage without an allocator");

  c10::Device device = storage->device();

  // 特殊情况：调整为0大小
  if (size_bytes == 0) {
    storage->set_data_ptr_noswap(at::DataPtr(nullptr, device));
    storage->set_nbytes(0);
    return;
  }

  // 设置正确的CUDA设备上下文
  c10::cuda::CUDAGuard guard(device.index());

  // 分配新内存
  at::DataPtr data = allocator->allocate(size_bytes);

  // 如果存在旧数据，复制到新内存
  if (storage->data_ptr()) {
    at::globalContext().lazyInitCUDA();

    // 异步复制数据 - 高效的CUDA内存管理
    C10_CUDA_CHECK(
        cudaMemcpyAsync(
            data.get(),                                  // 目标：新内存
            storage->data(),                             // 源：旧内存
            std::min(storage->nbytes(), size_bytes),     // 复制量：较小的值
            cudaMemcpyDeviceToDevice,                    // 复制类型：设备到设备
```

```
33              c10::cuda::getCurrentCUDAStream()));          // 使用当前CUDA流
34    }
35
36    // 用新的数据指针替换旧的
37    storage->set_data_ptr_noswap(std::move(data));
38    storage->set_nbytes(size_bytes);
39  }
```

rsub.Tensor, rsub.Tensor_out, rsub.Scalar, rsub.Scalar_out

redispatch 到 sub

代码块

```
1   aten/src/ATen/native/BinaryOps.cpp
2
3   // rsub.Scalar -> rsub.Tensor
4   Tensor rsub(const Tensor& self, const Scalar& other, const Scalar& alpha) {
5     return native::rsub(self, wrapped_scalar_tensor(other), alpha);
6   }
7   // rsub.Tensor -> sub.Tensor
8   Tensor rsub(const Tensor& self, const Tensor& other, const Scalar& alpha) {
9     return at::sub(other, self, alpha); // redispatch!
10  }
```

floor_divide, floor_divide.out, floor_divide_.Tensor, floor_divide.Scalar, floor_divide.Scalar_out, floor_divide_.Scalar

参考信息

文档：https://pytorch.org/docs/stable/generated/torch.floor_divide.html

参考实现：
https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/BinaryDivFloorKernel.cu

floor_divide.Scalar, floor_divide.Scalar_out, floor_divide_.Scalar 在 build/aten/src/ATen/RegisterCompositeExplicitAutograd.cpp 中注册

build/aten/src/ATen/RegisterCompositeExplicitAutograd.cpp

```
1   m.impl("floor_divide.Scalar",
2   TORCH_FN(wrapper_CompositeExplicitAutograd_Scalar_floor_divide));
3   m.impl("floor_divide.Scalar_out",
4   TORCH_FN(wrapper_CompositeExplicitAutograd_Scalar_out_floor_divide_out));
```

```
  m.impl("floor_divide_.Scalar",
  TORCH_FN(wrapper_CompositeExplicitAutograd_Scalar_floor_divide_));
```

在 aten/src/ATen/native/BinaryOps.cpp 中 redispatch 到 floor_divide

代码块

```
Tensor floor_divide(const Tensor& self, const Scalar& other) {
  return at::floor_divide(self, wrapped_scalar_tensor(other));
}

Tensor& floor_divide_(Tensor& self, const Scalar& other) {
  return at::floor_divide_out(self, self, wrapped_scalar_tensor(other));
}
```

通过最终 floor_divide 注册到 div_floor_stub

代码块

```
import torch

a = torch.tensor([5, 6, 7, 8])
b = torch.tensor([2, 2, 3, 3])

# 张量与张量整除
# 使用函数
result = torch.floor_divide(a, b)
print(result)  # tensor([2, 3, 2, 2])

# 使用运算符
result = a // b
print(result)  # tensor([2, 3, 2, 2])

# 张量方法
result = a.floor_divide(b)
print(result)  # tensor([2, 3, 2, 2])

a = torch.tensor([5, 6, 7, 8])

# 张量与标量整除
# 使用函数
result = torch.floor_divide(a, 2)
print(result)  # tensor([2, 3, 3, 4])

# 使用运算符
result = a // 2
```

```python
print(result)  # tensor([2, 3, 3, 4])

# 标量在左边
result = torch.floor_divide(2, a)
print(result)  # tensor([0, 0, 0, 0])  # 因为 2/5, 2/6 等都小于1，向下取整为0

# 广播支持
a = torch.tensor([[5, 6], [7, 8]])
b = torch.tensor([2, 3])

result = a // b
print(result)
# tensor([[2, 2],
#         [3, 2]])

#当操作数中有浮点数时，结果为浮点数:
a = torch.tensor([5.0, 6.0, 7.0])
b = torch.tensor([2.0, 2.0, 2.0])

# 普通除法
div_result = a / b
print(div_result)  # tensor([2.5000, 3.0000, 3.5000])

# 整除
floor_div_result = a // b
print(floor_div_result)  # tensor([2., 3., 3.])

# 等价于
floor_div_alternative = torch.floor(a / b)
print(floor_div_alternative)  # tensor([2., 3., 3.])
```

Function 接口

代码块

```
/**
 * @brief Performs element-wise floor division (a // b)
 *
 * @param dev             Device handle for CUDA/CPU device management
 * @param out             Output tensor to store the result
 * @param self            Dividend tensor (left operand)
 * @param other           Divisor tensor or scalar (right operand)
 * @param out_sizes       Output tensor sizes
 * @param out_strides     Output tensor strides (can be non-contiguous)
 * @param self_sizes      Dividend tensor sizes
 * @param self_strides    Dividend tensor strides (can be non-contiguous)
 * @param other_sizes     Divisor tensor sizes (nullptr if scalar)
```

```
 * @param other_strides    Divisor tensor strides (can be non-contiguous)
 * @param out_ndims        Number of output tensor dimensions
 * @param self_ndims       Number of dividend tensor dimensions
 * @param other_ndims      Number of divisor tensor dimensions
 * @param self_dtype       Data type of dividend tensor
 * @param other_dtype      Data type of divisor tensor
 * @param is_other_scalar  Whether other is a scalar
 *
 * @note Memory Layout:
 *       - Fully supports non-contiguous tensors
 *       - Handles arbitrary strides efficiently
 *       - Optimized for both contiguous and non-contiguous cases
 *
 * @note Behavior:
 *       - Performs division followed by floor operation
 *       - Handles type promotion according to PyTorch rules
 *       - Properly handles division by zero according to IEEE standards
 *       - For integer inputs, returns integer floor division (equivalent to Python //)
 */
void floor_divide_kernel(
    SiDeviceHandle *dev,           // Device handle
    void *out,                     // Output tensor data pointer
    const void *self,              // Dividend tensor data pointer
    const void *other,             // Divisor tensor/scalar data pointer
    const int64_t* out_sizes,      // Output sizes
    const int64_t* out_strides,    // Output strides
    const int64_t* self_sizes,     // Dividend tensor sizes
    const int64_t* self_strides,   // Dividend tensor strides
    const int64_t* other_sizes,    // Divisor tensor sizes (nullptr if scalar)
    const int64_t* other_strides,  // Divisor tensor strides (nullptr if scalar)
    int out_ndims,                 // Number of output dimensions
    int self_ndims,                // Number of dividend dimensions
    int other_ndims,               // Number of divisor dimensions
    int self_dtype,                // Data type of dividend
    int other_dtype,               // Data type of divisor
    bool is_other_scalar           // Whether divisor is a scalar
);

支持的数据类型
整数类型：uint8, int8, int16, int32, int64
浮点类型：float16 (half), bfloat16, float32, float64
```

首个版本开发说明

（1）数制支持上，初版需要支持int32类型，如果输入数据类型为int64，由框架进行int64->int32转换

（2）除数（输入2）需要支持tensor与scalar两种形式

（3）输出与被除数（输入1）的shape一致

（4）如果除数为tensor，那么除数与被除数shape应相同（如不同，则由框架处理进行broadcast），做elementwise除法

（5）对齐限制：全流程使用RVV Core，无对齐限制

（6）输入输出参数，均由框架进行预处理（reshape为一维）

内部接口调用形式：

代码块

```
/// @brief Tensor floor division
/// @tparam T Datatype support [int32_t]
/// @param[in] d_in0 Dividend tensor with shape [dim0_size]
/// @param[in] d_in1 Divisor tensor with shape [dim0_size] or scalar if is_d_in1_scalar is true
/// @param[out] d_out Output tensor with shape [dim0_size]
/// @param[in] dim0_size Tensor shape param
/// @param[in] is_d_in1_scalar Indicates d_in1 is scalar or tensor
template <typename T>
void floor_div(void *d_in0, void *d_in1, void *d_out, int dim0_size, bool is_d_in1_scalar);
```

ISSUE

- [ ] 非 contiguous 的是否能支持（如果支持需要修改 Function 接口）
  - [ ] pytorch 中使用了 TensorIteratorBase 传递 self，other， out， 需考虑非 contiguous 实现
- [ ] 考虑到 broadcast 的情况，out 和 self 的 size/stride 存在不一致的情况，需要传入
- [ ] Memory 是否需要 Align，Align 到多少

repeat_interleave.self_Tensor, repeat_interleave.self_int

参考信息

参考文档：https://pytorch.org/docs/stable/generated/torch.repeat_interleave.html

repeat_interleave 是由以下几个 op 组合完成：

is_conj, conj, is_neg, _neg_view, flatten, reshape, expand_symint, index_select, unsqueeze, clone

代码块

```
# 所有元素重复相同次数
>>> x = torch.tensor([1, 2, 3])
>>> repeated = torch.repeat_interleave(x, repeats=3)
>>> repeated
tensor([1, 1, 1, 2, 2, 2, 3, 3, 3])

# 指定维度重复
>>> x = torch.tensor([[1, 2], [3, 4]])
>>> repeated_rows = torch.repeat_interleave(x, repeats=2, dim=0)
>>> repeated_rows
tensor([[1, 2],
        [1, 2],
        [3, 4],
        [3, 4]])
>>> repeated_cols = torch.repeat_interleave(x, repeats=3, dim=1)
>>> repeated_cols
tensor([[1, 1, 1, 2, 2, 2],
        [3, 3, 3, 4, 4, 4]])

# 每个元素重复不同次数
>>> x = torch.tensor([1, 2, 3])
>>> repeats = torch.tensor([1, 2, 3])
>>> repeated = torch.repeat_interleave(x, repeats)
>>> repeated
tensor([1, 2, 2, 3, 3, 3])

# 在多维张量上使用不同的重复次数
>>> x = torch.tensor([[1, 2, 3], [4, 5, 6]])
>>> row_repeats = torch.tensor([1, 3])  # 第一行重复1次，第二行重复3次
>>> repeated_rows = torch.repeat_interleave(x, row_repeats, dim=0)
>>> repeated_rows
tensor([[1, 2, 3],
        [4, 5, 6],
        [4, 5, 6],
        [4, 5, 6]])
```

代码块

```
// repeat_interleave.self_Tensor
Tensor repeat_interleave_symint(
    const Tensor& self,
    const Tensor& repeats,
    std::optional<int64_t> dim,
    std::optional<SymInt> output_size) {
  Tensor input = self;

```

```
  // Store conj and neg bits
  const auto conj = input.is_conj();
  if (conj) {
    input = input.conj();
  }
  const auto neg = input.is_neg();
  if (neg) {
    input = input._neg_view();
  }

  if (!dim) {
    input = input.flatten();
    dim = 0;
  }

  Tensor repeats_ = repeats;
  if (repeats.dim() == 0 || (repeats.dim() == 1 && repeats.sym_size(0) == 1)) {
    repeats_ = repeats.reshape({1}).expand_symint({input.sym_size(dim.value())});
  } else if (repeats.dim() == 1) {
    TORCH_CHECK(
        repeats.sym_size(0) == input.sym_size(dim.value()),
        "repeats must have the same size as input along dim, but got repeats.size(0) = ",
        repeats.sym_size(0), " and input.size(", dim.value(), ") = ", input.sym_size(dim.value())
    );
  } else {
    AT_ERROR("repeats must be 0-dim or 1-dim tensor");
  }

  auto ret = input.index_select(
      dim.value(), at::repeat_interleave_symint(repeats_, std::move(output_size)));
  // Restore conj and neg bits
  if (conj) {
    ret = ret.conj();
  }
  if (neg) {
    ret = ret._neg_view();
  }
  return ret;
}

// repeat_interleave.self_int
Tensor repeat_interleave_symint(
    const Tensor& self,
```

```
52      c10::SymInt repeats,
53      std::optional<int64_t> dim_opt,
54      std::optional<SymInt> output_size) {
55    Tensor input = dim_opt ? self : self.flatten();
56    int64_t dim = c10::maybe_wrap_dim(dim_opt.value_or(0), self.dim());
57    TORCH_CHECK(repeats >= 0, "Repeats must be non-negative");
58
59    input = input.unsqueeze(dim + 1);
60    auto expand_shape = input.sym_sizes().vec();
61    expand_shape[dim + 1] = repeats;
62    input = input.expand_symint(expand_shape);
63
64    // This argument doesn't really make sense for the scalar overload, but
    exists
65    // for consistency with the tensor overload
66    if (output_size) {
67      auto calculated_size = (repeats * expand_shape[dim]).guard_int(__FILE__,
    __LINE__);
68      TORCH_CHECK(*output_size == calculated_size, "repeat_interleave: Invalid
    output_size, expected ",
69                  calculated_size, " but got ", *output_size);
70    }
71
72    return input.clone(at::MemoryFormat::Contiguous).flatten(dim, dim + 1);
73  }
74
```

addmv, addmv.out, addmv_

参考信息

pytorch官方文档：https://pytorch.org/docs/stable/generated/torch.addmv.html

Addmv 通过cuda cublas的gemv实现，参考cublas文档
https://docs.nvidia.com/cuda/cublas/#cublas-t-gemv

调用位置：
https://github.com/pytorch/pytorch/blob/164d2c887b45ec2673b9bbc08e457bfab0a1eb3c/aten/src/ATen/native/cuda/Blas.cpp#L839

代码块

```
1  >>> input = torch.tensor([1, 2, 3], dtype=torch.float)
2  >>> mat = torch.tensor([[2, 3, 4], [5, 6, 7], [8, 9, 10]], dtype=torch.float)
3  >>> vec = torch.tensor([1, 2, 3], dtype=torch.float)
```

```
>>> result = torch.addmv(input, mat, vec)
>>> result
tensor([21., 40., 59.])
>>> result = torch.addmv(input, mat, vec, beta=2, alpha=3)
>>> result
tensor([ 62., 118., 174.])
>>> manual = input + mat @ vec
>>> manual
tensor([21., 40., 59.])
>>> manual2 = 2 * input + 3 * (mat @ vec)
>>> manual2
tensor([ 62., 118., 174.])
```

Function 接口

代码块

```
/**
 * @brief Performs matrix-vector multiplication with scaling and addition (addmv operation)
 *
 * @param dev              Device handle for CUDA device management
 * @param out              Output tensor to store the result
 * @param self             Input vector to be scaled by beta
 * @param mat              Input matrix for matrix-vector multiplication
 * @param vec              Input vector for matrix-vector multiplication
 * @param beta             Scalar multiplier for self
 * @param alpha            Scalar multiplier for the matrix-vector product
 * @param out_sizes        Output tensor sizes
 * @param out_strides      Output tensor strides (can be non-contiguous)
 * @param self_sizes       Input vector sizes
 * @param self_strides     Input vector strides (can be non-contiguous)
 * @param mat_sizes        Matrix tensor sizes
 * @param mat_strides      Matrix tensor strides (can be non-contiguous)
 * @param vec_sizes        Vector tensor sizes
 * @param vec_strides      Vector tensor strides (can be non-contiguous)
 * @param out_ndims        Number of output tensor dimensions
 * @param self_ndims       Number of input vector dimensions
 * @param mat_ndims        Number of matrix tensor dimensions
 * @param vec_ndims        Number of vector tensor dimensions
 * @param self_dtype       Data type of input vector
 * @param mat_dtype        Data type of matrix tensor
 * @param vec_dtype        Data type of vector tensor
 *
 * @note Memory Layout:
 *       - Fully supports non-contiguous tensors
 *       - Optimized for different matrix layouts:
```

```
 *         - Row-major: Uses direct gemv with 'n' mode
 *         - Column-major: Uses transposed gemv with 't' mode
 *       - Handles arbitrary strides efficiently
 *
 * @note Behavior:
 *       - Computes: out = beta * self + alpha * (mat @ vec)
 *       - If mat is empty, either zeroes out or computes beta * self
 *       - Handles type promotion according to PyTorch rules
 *       - For vectors with stride 0, creates contiguous copies
 */
template <typename T>
void addmv_cuda_kernel(
    SiDeviceHandle *dev,           // Device handle
    void *out,                     // Output tensor data pointer
    const void *self,              // Input vector data pointer
    const void *mat,               // Matrix tensor data pointer
    const void *vec,               // Vector tensor data pointer
    const T& beta,                 // Scalar multiplier for self
    const T& alpha,                // Scalar multiplier for mat @ vec
    const int64_t* out_sizes,      // Output sizes
    const int64_t* out_strides,    // Output strides
    const int64_t* self_sizes,     // Input vector sizes
    const int64_t* self_strides,   // Input vector strides
    const int64_t* mat_sizes,      // Matrix tensor sizes
    const int64_t* mat_strides,    // Matrix tensor strides
    const int64_t* vec_sizes,      // Vector tensor sizes
    const int64_t* vec_strides,    // Vector tensor strides
    int out_ndims,                 // Number of output dimensions
    int self_ndims,                // Number of input vector dimensions
    int mat_ndims,                 // Number of matrix dimensions
    int vec_ndims,                 // Number of vector dimensions
    int self_dtype,                // Data type of input vector
    int mat_dtype,                 // Data type of matrix
    int vec_dtype                  // Data type of vector
);

支持的数据类型：
浮点类型：float, double
复数类型：complex<float>, complex<double>
特殊类型：Half, BFloat16
```

ISSUE

- [ ] cuda的实现支持非contiguous输入，也支持行优先/列优先模式

conj_physical.out

参考信息

pytorch官方文档：https://pytorch.org/docs/stable/generated/torch.conj_physical.html

参考实现：
https://github.com/pytorch/pytorch/blob/836955bdbdeb299e6937065299564fb44ec422c2/aten/src/ATen/native/cuda/UnaryComplexKernels.cu#L66

代码块

```
conj_physical 是 PyTorch 中用于处理复数张量的函数，它返回一个新的张量，其中每个复数元素都被替换为其共轭（conjugate）
对于复数类型（kComplexFloat、kComplexDouble、kComplexHalf），执行真正的共轭操作
对于所有其他类型（整数、浮点数、布尔值），执行直接复制操作（no-op）

>>> x = torch.tensor([1+2j, 3-4j, 5+0j])
>>> y = torch.conj_physical(x)
>>> x
tensor([1.+2.j, 3.-4.j, 5.+0.j])
>>> y
tensor([1.-2.j, 3.+4.j, 5.-0.j])
```

Function 接口

代码块

```
/**
 * @brief Computes the complex conjugate of each element in the input tensor
 *
 * @param dev             Device handle for CUDA device management
 * @param out             Output tensor to store the result
 * @param self            Input tensor
 * @param out_sizes       Output tensor sizes
 * @param out_strides     Output tensor strides (can be non-contiguous)
 * @param self_sizes      Input tensor sizes
 * @param self_strides    Input tensor strides (can be non-contiguous)
 * @param out_ndims       Number of output tensor dimensions
 * @param self_ndims      Number of input tensor dimensions
 * @param out_dtype       Data type of output tensor
 * @param self_dtype      Data type of input tensor
 *
 * @note Memory Layout:
 *       - Fully supports non-contiguous tensors
 *       - Handles arbitrary strides efficiently
 *       - Optimized for both contiguous and non-contiguous cases
```

```
 *
 * @note Behavior:
 *       - For complex types: returns the complex conjugate (a-bi for a+bi)
 *       - For non-complex types: performs a direct copy (no-op)
 *       - Preserves NaN values
 *       - Supports complex64, complex128, and complex half types
 *       - For boolean and real types, performs a direct copy
 *
 * @note Performance:
 *       - Uses JIT compilation when available for optimal performance
 *       - Falls back to explicit kernel implementation when JIT is unavailable
 *       - Optimized for parallel execution on CUDA devices
 */
template <typename T>
void conj_kernel_cuda(
    SiDeviceHandle *dev,           // Device handle
    void *out,                     // Output tensor data pointer
    const void *self,              // Input tensor data pointer
    const int64_t* out_sizes,      // Output sizes
    const int64_t* out_strides,    // Output strides
    const int64_t* self_sizes,     // Input tensor sizes
    const int64_t* self_strides,   // Input tensor strides
    int out_ndims,                 // Number of output dimensions
    int self_ndims,                // Number of input dimensions
    int out_dtype,                 // Data type of output
    int self_dtype                 // Data type of input
);

支持的数据类型
kByte (uint8_t)
kChar (int8_t)
kShort (int16_t)
kInt (int32_t)
kLong (int64_t)
所有浮点类型:
kFloat (float)
kDouble (double)
kHalf (at::Half)
kBFloat16 (at::BFloat16)
布尔类型:
kBool (bool)
复数类型
kComplexFloat (c10::complex<float>)
32位复数类型，由两个单精度浮点数组成
kComplexDouble (c10::complex<double>)
64位复数类型，由两个双精度浮点数组成
kComplexHalf (c10::complex<at::Half>)
```

```
16位复数类型，由两个半精度浮点数组成

```

ISSUE

- [ ] cuda中通过TensorIteratorBase来传递输入和输出，支持非contiguous
- [ ]

dot

参考信息

pytorch资料：https://pytorch.org/docs/stable/generated/torch.dot.html

cuda的实现使用的是cublas，参考资料：https://docs.nvidia.com/cuda/cublas/#cublas-t-dot

Computes the dot product of two 1D tensors.

Function 接口

代码块

```
/**
 * @brief Computes the dot product of two 1-D vectors
 *
 * @param dev             Device handle for CUDA device management
 * @param out             Output tensor to store the result (scalar)
 * @param self            First input vector
 * @param other           Second input vector
 * @param self_sizes      First vector sizes
 * @param other_sizes     Second vector sizes
 * @param self_strides    First vector strides (can be non-contiguous)
 * @param other_strides   Second vector strides (can be non-contiguous)
 * @param self_ndims      Number of dimensions in first vector (should be 1)
 * @param other_ndims     Number of dimensions in second vector (should be 1)
 * @param self_dtype      Data type of first vector
 * @param other_dtype     Data type of second vector
 *
 * @note Memory Layout:
 *       - Supports non-contiguous vectors with arbitrary strides
 *       - For vectors of size 1, strides are treated as 1 regardless of actual value
 *       - Optimized for both contiguous and non-contiguous cases
 *
 * @note Behavior:
 *       - Computes sum(self[i] * other[i]) for all i
```

```
 *       - For complex types, performs standard complex multiplication (not conjugated)
 *       - Returns a scalar tensor of the same type as inputs
 *       - If either input is a zero tensor, returns an efficient zero tensor
 *       - For complex inputs with conjugation flags, special handling is applied:
 *         - If self is conjugated but other is not: equivalent to vdot(self.conj(), other)
 *         - If other is conjugated but self is not: equivalent to vdot(other.conj(), self)
 *         - If both are conjugated: equivalent to dot(self.conj(), other.conj()).conj()
 *
 * @note Requirements:
 *       - Both vectors must have the same number of elements
 *       - Both vectors must have the same data type
 *       - Number of elements must be ≤ INT_MAX
 *       - Strides must be ≤ INT_MAX
 *
 * @note Performance:
 *       - Uses cuBLAS dot/dotc functions for optimal performance
 *       - Sets cuBLAS pointer mode to DEVICE for direct output to tensor
 */
template <typename T>
void dot_cuda_kernel(
    SiDeviceHandle *dev,           // Device handle
    void *out,                     // Output tensor data pointer (scalar)
    const void *self,              // First vector data pointer
    const void *other,             // Second vector data pointer
    const int64_t self_sizes,      // First vector sizes
    const int64_t other_sizes,     // Second vector sizes
    const int64_t self_strides,    // First vector strides
    const int64_t other_strides,   // Second vector strides
    int self_dtype,                // Data type of first vector
    int other_dtype,               // Data type of second vector
    bool self_is_conj,             // Whether first vector is conjugated
    bool other_is_conj             // Whether second vector is conjugated
);
```

支持的数据类型
浮点类型：
`float` (单精度浮点数)
`double` (双精度浮点数)
半精度类型：
`Half` (16位半精度浮点数)
`BFloat16` (16位脑浮点数)
复数类型：

```
complex<float> （复数，实部和虚部都是单精度浮点数）
complex<double> （复数，实部和虚部都是双精度浮点数）
```

ISSUE

- [ ] 支持非连续向量，需要明确处理步长

replication_pad1d, replication_pad1d.out

参考信息

pytorch文档：https://pytorch.org/docs/stable/generated/torch.nn.ReplicationPad1d.html

参考实现
https://github.com/pytorch/pytorch/blob/2e7c9d33e7f933ac3b723cb3bb05b9c88432c25c/aten/src/ATen/native/cuda/ReplicationPadding.cu#L419

代码块

```python
import torch
import torch.nn as nn

# 创建一个输入张量
input = torch.tensor([[[1, 2, 3, 4, 5]]], dtype=torch.float)

# 创建一个 ReplicationPad1d 层
pad_layer = nn.ReplicationPad1d(2) #左右均pad 2 个

# 应用填充
output = pad_layer(input)
print(output)
# 输出: tensor([[[1, 1, 1, 2, 3, 4, 5, 5, 5]]])

pad_layer = nn.ReplicationPad1d(1,2) # 左侧pad 1 右侧 pad 2
# 应用填充
output = pad_layer(input)
print(output)
# 输出: tensor([[[1., 1., 2., 3., 4., 5., 5., 5.]]])


m = nn.ReplicationPad1d(2)
input = torch.arange(8, dtype=torch.float).reshape(1, 2, 4)
# tensor([[[0., 1., 2., 3.],         [4., 5., 6., 7.]]])
m(input)
# tensor([[[0., 0., 0., 1., 2., 3., 3., 3.],
#         [4., 4., 4., 5., 6., 7., 7., 7.]]])

# using different paddings for different sides
```

```
m = nn.ReplicationPad1d((3, 1))
m(input)
# tensor([[[0., 0., 0., 0., 1., 2., 3., 3.],
#         [4., 4., 4., 4., 5., 6., 7., 7.]]])
```

Function 接口

代码块

```
/**
 * @brief Performs 1D replication padding on input tensor
 *
 * @param dev             Device handle for CUDA device management
 * @param out             Output tensor to store the result
 * @param input           Input tensor to be padded
 * @param padL            Number of padding elements on the left side
 * @param padR            Number of padding elements on the right side
 * @param out_sizes       Output tensor sizes
 * @param out_strides     Output tensor strides (can be non-contiguous)
 * @param input_sizes     Input tensor sizes
 * @param input_strides   Input tensor strides (can be non-contiguous)
 * @param out_ndims       Number of dimensions in output tensor
 * @param input_ndims     Number of dimensions in input tensor
 * @param input_dtype     Data type of input tensor
 *
 * @note Memory Layout:
 *       - Supports non-contiguous tensors with arbitrary strides
 *       - Handles both 2D (C, W) and 3D (N, C, W) input formats
 *       - For 2D inputs, the operation is applied to each channel independently
 *
 * @note Behavior:
 *       - Pads by replicating the edge values of the input tensor
 *       - For example, with input [1, 2, 3] and padding (2, 1):
 *         Output will be [1, 1, 1, 2, 3, 3]
 *       - Supports all data types including floating point, integer, and complex types
 *
 * @note Performance:
 *       - Uses CUDA kernel with block size optimization for different tensor sizes
 *       - Processes tensors in chunks to handle large batch sizes efficiently
 *       - Automatically handles tensor dimensions > 65535 by splitting into multiple kernel launches
 */
template <typename T>
void replication_pad1d_out_kernel(
```

```
    SiDeviceHandle *dev,          // Device handle
    void *out,                    // Output tensor data pointer
    const void *input,            // Input tensor data pointer
    int64_t padL,                 // Left padding size
    int64_t padR,                 // Right padding size
    const int64_t* out_sizes,     // Output sizes
    const int64_t* out_strides,   // Output strides
    const int64_t* input_sizes,   // Input tensor sizes
    const int64_t* input_strides, // Input tensor strides
    int out_ndims,                // Number of output dimensions
    int input_ndims,              // Number of input dimensions
    int input_dtype               // Data type of input
);

新接口:
template <typename T>
void replication_pad_out_kernel(
    const sikernel::Tensor<T>& self,
    sikernel::Tensor<T>& out,
    const void* padding,
    int padding_arr_size,
);

支持数据类型:
整数类型: uint8_t, int8_t, int16_t, int32_t, int64_t
浮点类型: float, double
半精度类型: Half (kHalf), BFloat16 (kBFloat16)
复数类型: complex<float>, complex<double>
```

ISSUE

- [ ] 支持非contiguous，同时支持2D,3D输入
- [ ] cuda的实现通过分块处理超过 CUDA 网格限制（65535）的大型张量

replication_pad2d, replication_pad2d.out

参考信息

pytorch文档：https://pytorch.org/docs/stable/generated/torch.nn.ReplicationPad2d.html

参考实现：
https://github.com/pytorch/pytorch/blob/2e7c9d33e7f933ac3b723cb3bb05b9c88432c25c/aten/src/ATen/native/cuda/ReplicationPadding.cu#L534

代码块

```
>>> m = nn.ReplicationPad2d(2)
```

```
>>> input = torch.arange(9, dtype=torch.float).reshape(1, 1, 3, 3)
>>> input
tensor([[[[0., 1., 2.],
          [3., 4., 5.],
          [6., 7., 8.]]]])
>>> m(input)
tensor([[[[0., 0., 0., 1., 2., 2., 2.],
          [0., 0., 0., 1., 2., 2., 2.],
          [0., 0., 0., 1., 2., 2., 2.],
          [3., 3., 3., 4., 5., 5., 5.],
          [6., 6., 6., 7., 8., 8., 8.],
          [6., 6., 6., 7., 8., 8., 8.],
          [6., 6., 6., 7., 8., 8., 8.]]]])

>>> m = nn.ReplicationPad2d((1, 1, 2, 0)) #  (padding_lef, padding_right, padding_top, padding_bottom)
>>> m(input)
tensor([[[[0., 0., 1., 2., 2.],
          [0., 0., 1., 2., 2.],
          [0., 0., 1., 2., 2.],
          [3., 3., 4., 5., 5.],
          [6., 6., 7., 8., 8.]]]])
>>>
```

Function 接口

代码块

```
/**
 * @brief Performs 2D replication padding on input tensor
 *
 * @param dev              Device handle for CUDA device management
 * @param out              Output tensor to store the result
 * @param input            Input tensor to be padded
 * @param padL             Number of padding elements on the left side
 * @param padR             Number of padding elements on the right side
 * @param padT             Number of padding elements on the top side
 * @param padB             Number of padding elements on the bottom side
 * @param out_sizes        Output tensor sizes
 * @param out_strides      Output tensor strides (can be non-contiguous)
 * @param input_sizes      Input tensor sizes
 * @param input_strides    Input tensor strides (can be non-contiguous)
 * @param out_ndims        Number of dimensions in output tensor
 * @param input_ndims      Number of dimensions in input tensor
 * @param input_dtype      Data type of input tensor
 *
 * @note Memory Layout:
```

```
 *       - Supports non-contiguous tensors with arbitrary strides
 *       - Handles both 3D (C, H, W) and 4D (N, C, H, W) input formats
 *       - For 3D inputs, the operation is applied to each channel independently
 *
 * @note Behavior:
 *       - Pads by replicating the edge values of the input tensor
 *       - For example, with a 3x3 input and padding (1,1,1,1), the corner values
 *         are replicated diagonally and edge values are replicated outward
 *       - Supports all data types including floating point, integer, and complex types
 *
 * @note Performance:
 *       - Uses CUDA kernel with block size optimization for different tensor sizes
 *       - Processes tensors in chunks to handle large batch sizes efficiently
 *       - Automatically handles tensor dimensions > 65535 by splitting into multiple kernel launches
 *       - Optimized for both small and large padding sizes
 *
 * @note Requirements:
 *       - Input tensor must fit into 32-bit index math
 *       - Output tensor must have correct size: [N, C, H+padT+padB, W+padL+padR]
 */
 和replication_pad1d接口一样
template <typename T>
void replication_pad_out_kernel(
    const sikernel::Tensor<T>& self,
    sikernel::Tensor<T>& out,
    const void* padding,
    int padding_arr_size,
);
```

ISSUE

- [ ] 支持非contiguous，同时支持3D,4D输入
- [ ] cuda的实现通过分块处理超过 CUDA 网格限制（65535）的大型张量
- [ ] cuda的实现没有接收padR，padB。而是通过padL和padT计算出原始数据的起始位置进行填充，然后再根据output的大小，在padding部分直接复制最近的边界值

replication_pad3d, replication_pad3d.out

参考信息

参考信息

pytorch文档：https://pytorch.org/docs/stable/generated/torch.nn.ReplicationPad3d.html

参考实现：
https://github.com/pytorch/pytorch/blob/2e7c9d33e7f933ac3b723cb3bb05b9c88432c25c/aten/src/ATen/native/cuda/ReplicationPadding.cu#L602

代码块

```
>>> m = nn.ReplicationPad3d(1)
>>> input = torch.randn(2, 2, 2, 2, 2)
>>> output = m(input)
>>> input
tensor([[[[[ 0.1190, -2.7971],
           [ 0.3847, -0.9118]],

          [[ 0.7726, -0.1665],
           [-0.2704, -0.9113]]],


         [[[-0.1435, -2.3199],
           [-0.3886,  0.3390]],

          [[-1.7399,  0.3719],
           [ 1.6875,  0.3726]]]],


        [[[[ 0.1996, -0.6908],
           [-0.3071, -1.2073]],

          [[ 1.0142, -0.7493],
           [ 1.9937, -0.4326]]],


         [[[ 0.4510, -0.8870],
           [ 0.7344, -1.1570]],

          [[-0.7517,  0.7564],
           [-0.1884, -0.0280]]]]])
>>> output
tensor([[[[[ 0.1190,  0.1190, -2.7971, -2.7971],
           [ 0.1190,  0.1190, -2.7971, -2.7971],
           [ 0.3847,  0.3847, -0.9118, -0.9118],
           [ 0.3847,  0.3847, -0.9118, -0.9118]],

          [[ 0.1190,  0.1190, -2.7971, -2.7971],
```

```
            [ 0.1190,  0.1190, -2.7971, -2.7971],
            [ 0.3847,  0.3847, -0.9118, -0.9118],
            [ 0.3847,  0.3847, -0.9118, -0.9118]],

           [[ 0.7726,  0.7726, -0.1665, -0.1665],
            [ 0.7726,  0.7726, -0.1665, -0.1665],
            [-0.2704, -0.2704, -0.9113, -0.9113],
            [-0.2704, -0.2704, -0.9113, -0.9113]],

           [[ 0.7726,  0.7726, -0.1665, -0.1665],
            [ 0.7726,  0.7726, -0.1665, -0.1665],
            [-0.2704, -0.2704, -0.9113, -0.9113],
            [-0.2704, -0.2704, -0.9113, -0.9113]]],


          [[[-0.1435, -0.1435, -2.3199, -2.3199],
            [-0.1435, -0.1435, -2.3199, -2.3199],
            [-0.3886, -0.3886,  0.3390,  0.3390],
            [-0.3886, -0.3886,  0.3390,  0.3390]],

           [[-0.1435, -0.1435, -2.3199, -2.3199],
            [-0.1435, -0.1435, -2.3199, -2.3199],
            [-0.3886, -0.3886,  0.3390,  0.3390],
            [-0.3886, -0.3886,  0.3390,  0.3390]],

           [[-1.7399, -1.7399,  0.3719,  0.3719],
            [-1.7399, -1.7399,  0.3719,  0.3719],
            [ 1.6875,  1.6875,  0.3726,  0.3726],
            [ 1.6875,  1.6875,  0.3726,  0.3726]],

           [[-1.7399, -1.7399,  0.3719,  0.3719],
            [-1.7399, -1.7399,  0.3719,  0.3719],
            [ 1.6875,  1.6875,  0.3726,  0.3726],
            [ 1.6875,  1.6875,  0.3726,  0.3726]]]],


         [[[[ 0.1996,  0.1996, -0.6908, -0.6908],
            [ 0.1996,  0.1996, -0.6908, -0.6908],
            [-0.3071, -0.3071, -1.2073, -1.2073],
            [-0.3071, -0.3071, -1.2073, -1.2073]],

           [[ 0.1996,  0.1996, -0.6908, -0.6908],
            [ 0.1996,  0.1996, -0.6908, -0.6908],
            [-0.3071, -0.3071, -1.2073, -1.2073],
            [-0.3071, -0.3071, -1.2073, -1.2073]],
```

```
           [[ 1.0142,  1.0142, -0.7493, -0.7493],
            [ 1.0142,  1.0142, -0.7493, -0.7493],
            [ 1.9937,  1.9937, -0.4326, -0.4326],
            [ 1.9937,  1.9937, -0.4326, -0.4326]],

           [[ 1.0142,  1.0142, -0.7493, -0.7493],
            [ 1.0142,  1.0142, -0.7493, -0.7493],
            [ 1.9937,  1.9937, -0.4326, -0.4326],
            [ 1.9937,  1.9937, -0.4326, -0.4326]]],


          [[[ 0.4510,  0.4510, -0.8870, -0.8870],
            [ 0.4510,  0.4510, -0.8870, -0.8870],
            [ 0.7344,  0.7344, -1.1570, -1.1570],
            [ 0.7344,  0.7344, -1.1570, -1.1570]],

           [[ 0.4510,  0.4510, -0.8870, -0.8870],
            [ 0.4510,  0.4510, -0.8870, -0.8870],
            [ 0.7344,  0.7344, -1.1570, -1.1570],
            [ 0.7344,  0.7344, -1.1570, -1.1570]],

           [[-0.7517, -0.7517,  0.7564,  0.7564],
            [-0.7517, -0.7517,  0.7564,  0.7564],
            [-0.1884, -0.1884, -0.0280, -0.0280],
            [-0.1884, -0.1884, -0.0280, -0.0280]],

           [[-0.7517, -0.7517,  0.7564,  0.7564],
            [-0.7517, -0.7517,  0.7564,  0.7564],
            [-0.1884, -0.1884, -0.0280, -0.0280],
            [-0.1884, -0.1884, -0.0280, -0.0280]]]])
>>>
```

Function 接口

代码块

```
/**
 * @brief Performs 3D replication padding on input tensor
 *
 * @param dev             Device handle for CUDA device management
 * @param out             Output tensor to store the result
 * @param input           Input tensor to be padded
 * @param pfront          Number of padding elements on the front side
 * @param pback           Number of padding elements on the back side
 * @param ptop            Number of padding elements on the top side
 * @param pbottom         Number of padding elements on the bottom side
 * @param pleft           Number of padding elements on the left side
```

```
 * @param pright           Number of padding elements on the right side
 * @param out_sizes        Output tensor sizes
 * @param out_strides      Output tensor strides (can be non-contiguous)
 * @param input_sizes      Input tensor sizes
 * @param input_strides    Input tensor strides (can be non-contiguous)
 * @param out_ndims        Number of dimensions in output tensor
 * @param input_ndims      Number of dimensions in input tensor
 * @param input_dtype      Data type of input tensor
 *
 * @note Memory Layout:
 *       - Supports non-contiguous tensors with arbitrary strides
 *       - Handles both 4D (C, D, H, W) and 5D (N, C, D, H, W) input formats
 *       - For 4D inputs, the operation is applied to each channel independently
 *
 * @note Behavior:
 *       - Pads by replicating the edge values of the input tensor in all three dimensions
 *       - Edge values are replicated outward, and corner values are replicated diagonally
 *       - Supports all data types including floating point, integer, and complex types
 *
 * @note Performance:
 *       - Uses CUDA kernel with block size optimization for different tensor sizes
 *       - Processes tensors in chunks to handle large batch sizes efficiently
 *       - Automatically handles tensor dimensions > 65535 by splitting into multiple kernel launches
 *       - Optimized for both small and large padding sizes
 *
 * @note Requirements:
 *       - Input tensor must fit into 32-bit index math
 *       - Output tensor must have correct size: [N, C, D+pfront+pback, H+ptop+pbottom, W+pleft+pright]
 *       - At least one of the output dimensions (D, H, W) must be >= 1
 */
 和replication_pad1d接口一样
template <typename T>
void replication_pad_out_kernel(
    const sikernel::Tensor<T>& self,
    sikernel::Tensor<T>& out,
    const void* padding,
    int padding_arr_size,
);
```

ISSUE

- [ ] 支持非contiguous，同时支持4D，5D输入
- [ ] cuda的实现通过分块处理超过 CUDA 网格限制（65535）的大型张量
- [ ] cuda的实现没有接收*pright*，*pbottom*，*pback*。而是通过pleft，ptop和pfront计算出原始数据的起始位置进行填充，然后再根据output的大小，在padding部分直接复制最近的边界值

reflection_pad1d, reflection_pad1d.out

Pytorch 文档

- https://pytorch.org/docs/stable/generated/torch.nn.ReflectionPad1d.html

Pytorch 实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/ReflectionPad.cu#L430
- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/ReflectionPad.cu#L96
- tensor dim 数量必须为 2或者3：https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/PadNd.cpp#L205
- pad_l/pad_r 约束：Padding size should be less than the corresponding input dimension

接口：两个 tensor + pad_l + pad_r

代码块

```
Tensor & reflection_pad1d_out(const Tensor & self, c10::SymIntArrayRef padding, Tensor & out); // {"schema": "aten::reflection_pad1d.out(Tensor self, SymInt[2] padding, *, Tensor(a!) out) -> Tensor(a!)", "dispatch": "True", "default": "False"}
Tensor reflection_pad1d(const Tensor & self, c10::SymIntArrayRef padding); // {"schema": "aten::reflection_pad1d(Tensor self, SymInt[2] padding) -> Tensor", "dispatch": "True", "default": "True"}
```

代码块

```
m = torch.nn.ReflectionPad1d(5)
input = torch.arange(8, dtype=torch.float).reshape(1, 4, 2)
print(input)
print(m(input))
=> RuntimeError: Argument #4: Padding size should be less than the corresponding input dimension, but got: padding (5, 5) at dimension 2 of input
```

```
[1, 4, 2]
```

Function 接口

代码块

```
template <typename T>
void reflection_pad_out(const sikernel::Tensor<T>& self, sikernel::Tensor<T>&
out, const void* padding, int padding_arr_size);
```

reflection_pad2d,reflection_pad2d.out

Pytorch 文档

- https://pytorch.org/docs/stable/generated/torch.nn.ReflectionPad3d.html

Pytorch 实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/ReflectionPad.cu#L271
- Tensor dim 数量必须为 3或者4：https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/PadNd.cpp#L212
- 约束：Padding size should be less than the corresponding input dimension

接口：两个 tensor + padding_left, padding_right, padding_top, padding_bottom

对应关系：

Shape:

- Input: (N, C, H_{in}, W_{in}) or (C, H_{in}, W_{in}).
- Output: (N, C, H_{out}, W_{out}) or (C, H_{out}, W_{out}) where

$$H_{out} = H_{in} + \text{padding\_top} + \text{padding\_bottom}$$
$$W_{out} = W_{in} + \text{padding\_left} + \text{padding\_right}$$

代码块

```
Tensor & reflection_pad2d_out(const Tensor & self, c10::SymIntArrayRef
padding, Tensor & out); // {"schema": "aten::reflection_pad2d.out(Tensor self,
SymInt[4] padding, *, Tensor(a!) out) -> Tensor(a!)", "dispatch": "True",
"default": "False"}
Tensor reflection_pad2d(const Tensor & self, c10::SymIntArrayRef padding); //
{"schema": "aten::reflection_pad2d(Tensor self, SymInt[4] padding) -> Tensor",
"dispatch": "True", "default": "False"}
```

```
m = torch.nn.ReflectionPad2d(3)
input = torch.arange(8, dtype=torch.float).reshape(1, 2, 4)
print(input)
print(m(input))
=> RuntimeError: Argument #6: Padding size should be less than the corresponding input dimension, but got: padding (3, 3) at dimension 1 of input [1, 2, 4]
```

Fuction接口

同reflection_pad1d，接口复用。

reflection_pad3d, reflection_pad3d.out

Pytorch 文档

- https://pytorch.org/docs/stable/generated/torch.nn.ReflectionPad3d.html

Pytorch 实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/ReflectionPad.cu#L570
- Tensor dim 数量必须为 4或者5：https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/PadNd.cpp#L219
- 约束：Padding size should be less than the corresponding input dimension

接口：两个 tensor + padding_left, padding_right, padding_top, padding_bottom, padding_front, padding_back

对应关系：

Shape:

- Input: $(N, C, D_{in}, H_{in}, W_{in})$ or $(C, D_{in}, H_{in}, W_{in})$.
- Output: $(N, C, D_{out}, H_{out}, W_{out})$ or $(C, D_{out}, H_{out}, W_{out})$, where

$$D_{out} = D_{in} + \text{padding\_front} + \text{padding\_back}$$
$$H_{out} = H_{in} + \text{padding\_top} + \text{padding\_bottom}$$
$$W_{out} = W_{in} + \text{padding\_left} + \text{padding\_right}$$

代码块

```
Tensor & reflection_pad3d_out(const Tensor & self, c10::SymIntArrayRef padding, Tensor & out); // {"schema": "aten::reflection_pad3d.out(Tensor self, SymInt[6] padding, *, Tensor(a!) out) -> Tensor(a!)", "dispatch": "True", "default": "False"}
```

```
2   Tensor reflection_pad3d(const Tensor & self, c10::SymIntArrayRef padding); //
    {"schema": "aten::reflection_pad3d(Tensor self, SymInt[6] padding) -> Tensor",
    "dispatch": "True", "default": "True"}
```

代码块

```
1   m = torch.nn.ReflectionPad3d(3)
2   input = torch.arange(32, dtype=torch.float).reshape(1, 1, 2, 4, 4)
3   print(input)
4   print(m(input))
5
6   RuntimeError: Argument #8: Padding size should be less than the corresponding
    input dimension, but got: padding (3, 3) at dimension 2 of input [1, 1, 2, 4,
    4]
```

Fuction接口

同reflection_pad1d，接口复用。

scatter_.src, scatter.src, scatter.src_out

Pytorch 文档

- https://pytorch.org/docs/stable/generated/torch.Tensor.scatter_.html#torch.Tensor.scatter_ 的上半部分

Pytorch 实现

- 非 deterministic
- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/ScatterGatherKernel.cu#L477
- index dim 数量与 src 相同，但 sizes 不要求和 src 相同，详见 pytorch 文档

代码块

```
1   Tensor scatter(const Tensor & self, int64_t dim, const Tensor & index, const
    Tensor & src); // {"schema": "aten::scatter.src(Tensor self, int dim, Tensor
    index, Tensor src) -> Tensor", "dispatch": "True", "default": "True"}
2   Tensor & scatter_(Tensor & self, int64_t dim, const Tensor & index, const
    Tensor & src); // {"schema": "aten::scatter_.src(Tensor(a!) self, int dim,
    Tensor index, Tensor src) -> Tensor(a!)", "dispatch": "True", "default":
    "True"}
3   Tensor & scatter_out(const Tensor & self, int64_t dim, const Tensor & index,
    const Tensor & src, Tensor & out); // {"schema": "aten::scatter.src_out(Tensor
```

```
    self, int dim, Tensor index, Tensor src, *, Tensor(a!) out) -> Tensor(a!)",
    "dispatch": "True", "default": "False"}
```

和 gather 为互逆操作

代码块

```python
import torch
self = torch.zeros(3, 3, dtype=torch.int64)
src = torch.tensor([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])
index = torch.tensor([
    [0, 2, 1],
    [1, 0, 2],
    [2, 1, 0]
])
self.scatter_(dim=1, index=index, src=src)
new_src = self.gather(dim=1, index=index)
# src == new_src
```

Dtypes

- AT_ALL_TYPES: `uint8_t, int8_t, int16_t, int, int64_t, float, double`
- AT_COMPLEX_TYPES: `c10::complex<float>, c10::complex<double>`
- Half, Bool, BFloat16: `float16, bool, bfloat16`

代码块

```
AT_DISPATCH_ALL_TYPES_AND_COMPLEX_AND3(
    at::ScalarType::Half, at::ScalarType::Bool, at::ScalarType::BFloat16
```

scatter_.value, scatter.value, scatter.value_out

Pytorch 文档

- https://pytorch.org/docs/2.5/generated/torch.Tensor.scatter_.html#torch.Tensor.scatter_ 的下半部分

Pytorch 实现

- deterministic

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/ScatterGatherKernel.cu#L485

代码块

```
Tensor scatter(const Tensor & self, int64_t dim, const Tensor & index, const Scalar & value); // {"schema": "aten::scatter.value(Tensor self, int dim, Tensor index, Scalar value) -> Tensor", "dispatch": "True", "default": "True"}
Tensor & scatter_(Tensor & self, int64_t dim, const Tensor & index, const Scalar & value); // {"schema": "aten::scatter_.value(Tensor(a!) self, int dim, Tensor index, Scalar value) -> Tensor(a!)", "dispatch": "True", "default": "True"}
Tensor & scatter_out(const Tensor & self, int64_t dim, const Tensor & index, const Scalar & value, Tensor & out); // {"schema": "aten::scatter.value_out(Tensor self, int dim, Tensor index, Scalar value, *, Tensor(a!) out) -> Tensor(a!)", "dispatch": "True", "default": "False"}
```

Dtypes

- AT_ALL_TYPES: `uint8_t, int8_t, int16_t, int, int64_t, float, double`
- AT_COMPLEX_TYPES: `c10::complex<float>, c10::complex<double>`
- Half, Bool, BFloat16: `float16, bool, bfloat16`

代码块

```
AT_DISPATCH_ALL_TYPES_AND_COMPLEX_AND3(
      at::ScalarType::Half, at::ScalarType::Bool, at::ScalarType::BFloat16
```

sum, sum.dim_IntList, sum.IntList_out

参考信息

文档：https://pytorch.org/docs/stable/generated/torch.sum.html

参考实现：
https://github.com/pytorch/pytorch/blob/32f585d9346e316e554c8d9bf7548af9f62141fc/aten/src/ATen/native/cuda/ReduceSumProdKernel.cu#L173

注册到 sum_stub

代码块

```
import torch

# 基本用法 - torch.sum
# 创建一个示例张量
```

```python
x = torch.tensor([[1, 2, 3], [4, 5, 6]], dtype=torch.float)

# 对整个张量求和
total_sum = torch.sum(x)
print(f"整个张量的和: {total_sum}")  # tensor(21.)

# 沿着指定维度求和
row_sum = torch.sum(x, dim=0)  # 沿行方向求和（合并行）
print(f"行求和结果: {row_sum}")  # tensor([5., 7., 9.])

col_sum = torch.sum(x, dim=1)  # 沿列方向求和（合并列）
print(f"列求和结果: {col_sum}")  # tensor([6., 15.])

# 使用keepdim=True保持维度
row_sum_keep = torch.sum(x, dim=0, keepdim=True)
print(f"保持维度的行求和: {row_sum_keep}")  # tensor([[5., 7., 9.]])
print(f"形状: {row_sum_keep.shape}")  # torch.Size([1, 3])

# 多维度求和 - sum.dim_IntList
# 创建一个3D张量
y = torch.tensor([[[1, 2], [3, 4]],
                  [[5, 6], [7, 8]]], dtype=torch.float)
print(f"原始形状: {y.shape}")  # torch.Size([2, 2, 2])

# 沿多个维度求和 (dim=[0, 2])
multi_dim_sum = torch.sum(y, dim=[0, 2])
print(f"沿维度[0,2]的求和: {multi_dim_sum}")  # tensor([14., 22.])
print(f"结果形状: {multi_dim_sum.shape}")  # torch.Size([2])

# 保持维度
multi_dim_sum_keep = torch.sum(y, dim=[0, 2], keepdim=True)
print(f"保持维度的多维求和: {multi_dim_sum_keep}")
# tensor([[[14.],
#          [22.]]])
print(f"结果形状: {multi_dim_sum_keep.shape}")  # torch.Size([1, 2, 1])


```

Function 接口

代码块

```cpp
/**
 * @brief Performs tensor summation along specified dimensions
 *
 * @param dev               Device handle for CUDA/CPU device management
 * @param out               Output tensor to store the sum result
```

```
 * @param self              Input tensor to be summed
 * @param out_sizes         Output tensor sizes
 * @param out_strides       Output tensor strides (can be non-contiguous)
 * @param self_sizes        Input tensor sizes
 * @param self_strides      Input tensor strides (can be non-contiguous)
 * @param dim               Array of dimensions to reduce along
 * @param out_ndims         Number of dimensions to reduce (length of dims array)
 * @param self_ndims        Number of dimensions in the input tensor
 * @param dim_ndims         Number of dimensions in the input tensor
 * @param out_dtype         Data type of output tensor
 * @param input_dtype       Data type of input tensor
 * @param keepdim           Whether to keep the reduced dimensions with size 1
 *
 * @note Memory Layout:
 *       - Fully supports non-contiguous tensors
 *       - Handles arbitrary strides efficiently
 *       - Optimized for both contiguous and non-contiguous cases
 *
 * @note Behavior:
 *       - Accumulates sum in higher precision for floating point types
 *       - Performs efficient reduction using appropriate algorithms for each device
 *       - Handles empty tensors and edge cases correctly
 *       - Supports type promotion from input to output if needed
 */
void sum_kernel(
    SiDeviceHandle *dev,           // Device handle
    void *out,                     // Output tensor data pointer
    const void *self,              // Input tensor data pointer
    const int64_t* out_sizes,      // Output sizes
    const int64_t* out_strides,    // Output strides
    const int64_t* self_sizes,     // Input tensor sizes
    const int64_t* self_strides ,  // Input tensor strides
    const int64_t* dim,            // Dimensions to sum over
    int out_ndims,                 // Number of dimensions in input tensor
    int self_ndims,                // Number of dimensions in input tensor
    int dim_ndims,                 // Number of dimensions to reduce
    int out_dtype,                 // Data type of output tensor
    int input_dtype,               // Data type of input tensor
    bool keepdim                   // Whether to keep reduced dimensions as size 1
);
支持的数据类型
整数类型：int8, uint8, int16, int32, int64
浮点类型：float16, bfloat16, float32, float64
布尔类型：bool
```

```
复数类型：complex64，complex128
复数半精度：complex32
```

首个版本开发说明

（1）初版只需要支持dim=-1，keepdim=true的参数配置

（2）keepdim相当于unsqueeze操作，框架可以处理，算子不需要做相关适配

（3）数制支持上，初版只需要支持float类型

（4）初版可以只考虑tensor为continuous的情况

（5）执行流程为：torch将输入tensor（shape为[A, B, C]）进行shape变换，转为[A * B, C]传入sikernel，sikernel对最后一维进行累加计算，输出shape为[A * B]的tensor，然后torch将其还原到[A, B, 1]的shape形式

内部接口调用形式：

代码块

```
template <typename T>
void sum(void *d_in, void *d_out, int dim0_size, int dim1_size, int dim = -1);
```

备注：

（1）初版dim1_size参数，可实现为1024字节对齐（256个float），配合tile reg的load

实际开发情况，可将对齐要求缩减至32字节对齐（8个float）

ISSUE

- [ ] Cuda reduce sum kernel 通过并行处理和多级归约来实现，如果使用 sum_stub 来获取参数，此时只能获得 input 以及 output 的 shape/stride。上面接口提供的是完整参数列表，根据具体 sum sipu 实现，可以筛选其中一部分的参数作为接口使用
- [ ] 接受非 contiguous

zero_

Pytorch 文档： https://pytorch.org/docs/stable/generated/torch.Tensor.zero_.html

代码块

```
Tensor& zero_(Tensor &self) {
  int64_t nelements = c10::multiply_integers(self.sizes());
  if (self.device() == at::kCPU &&
      self.is_non_overlapping_and_dense() &&
      nelements < internal::GRAIN_SIZE) {
    return zero_cpu_(self, nelements);
```

```
  }
  return self.fill_(0);
}

非cpu backend会直接调用fill_


import torch

# 创建一个张量
x = torch.ones(3, 4)
print(x)
tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]])

# 将所有元素设置为零
x.zero_()
print(x)
tensor([[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]])
```

代码块

```
void zero_(SiDeviceHandle dev, void const* self, int dims, const int64_t*
self_sizes, const int64_t* self_strides, dtype)

```

topk, topk.values

Pytorch 文档

- https://pytorch.org/docs/stable/generated/torch.topk.html

Pytorch 实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/TensorTopK.cpp#L47
- 特定情况下直接 sort+narrow，其他情况下实现看起来是基于基数排序实现的（具体实现细节未细看）。依赖 cub 相关 apis。
- indices 默认 dtype 为 int64

```
::std::tuple<Tensor &,Tensor &> topk_out(const Tensor & self, c10::SymInt k, int64_t dim, bool largest, bool sorted, Tensor & values, Tensor & indices); // {"schema": "aten::topk.values(Tensor self, SymInt k, int dim=-1, bool largest=True, bool sorted=True, *, Tensor(a!) values, Tensor(b!) indices) -> (Tensor(a!) values, Tensor(b!) indices)", "dispatch": "True", "default": "False"}
::std::tuple<Tensor,Tensor> topk(const Tensor & self, c10::SymInt k, int64_t dim, bool largest, bool sorted); // {"schema": "aten::topk(Tensor self, SymInt k, int dim=-1, bool largest=True, bool sorted=True) -> (Tensor values, Tensor indices)", "dispatch": "True", "default": "True"}
```

Dtypes

- AT_ALL_TYPES: `uint8_t, int8_t, int16_t, int, int64_t, float, double`
- Half, BFloat16: `float16, bfloat16`

代码块

```
AT_DISPATCH_ALL_TYPES_AND2(at::ScalarType::Half, at::ScalarType::BFloat16
```

接口

- 在调用 kernel 前保证 contiguous，因此不传 strides

代码块

```
/**
 * @brief Compute the Top-K elements (values and indices) along a specified dimension of a tensor.
 *
 * @details
 * Extracts the top K largest/smallest elements from the input tensor along the given dimension,
 * writing results to separate value and index buffers. Supports both sorted and unsorted output.
 *
 * @param[in]  dev         Device handle (e.g., GPU/accelerator) for computation
 * @param[out] values      Output buffer for top K values (shape: original tensor with target dim set to K)
 * @param[out] indices     Output buffer for original indices of top K values (int64_t data type)
 * @param[in]  self        Input tensor data pointer (device memory)
 * @param[in]  dims        Number of dimensions in the input tensor
 * @param[in]  self_sizes  Array of dimension sizes (length = dims)
```

```
14    * @param[in]  k           Number of elements to select (0 < k <=
      self_sizes[dim])
15    * @param[in]  dim         Target dimension index (0-based, 0 <= dim < dims)
16    * @param[in]  largest     Select largest elements if true, smallest otherwise
17    * @param[in]  sorted      Sort results in descending order if true, preserve
      original order otherwise
18    *
19    */
20   void topk(SiDeviceHandle dev, void* values, void* indices, const void* self,
      int dims, const int64_t* self_sizes, int k, int dim, bool largest, bool sorted)
21
22   void topk(void* values, void* indices, const void* self, const int64_t
      batch_size, const int64_t size, int k, bool largest, bool sorted)
```

sigmoid.out, sigmoid, sigmoid_

pytorch 文档

- https://pytorch.org/docs/stable/generated/torch.nn.Sigmoid.html

pytorch 实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/UnarySpecialOpsKernel.cu#L124
- 使用 gpu_kernel + element-wise lambda function 实现

Unary op，接口定义同 neg

代码块

```
1   Tensor sigmoid(const Tensor & self); // {"schema": "aten::sigmoid(Tensor self)
    -> Tensor", "dispatch": "True", "default": "True"}
2   Tensor & sigmoid_(Tensor & self); // {"schema": "aten::sigmoid_(Tensor(a!)
    self) -> Tensor(a!)", "dispatch": "True", "default": "True"}
3   Tensor & sigmoid_out(const Tensor & self, Tensor & out); // {"schema":
    "aten::sigmoid.out(Tensor self, *, Tensor(a!) out) -> Tensor(a!)", "dispatch":
    "True", "default": "False"}
```

Dtypes

- AT_FLOATING_TYPES: `float, double`
- Half, BFloat16: `float16, bfloat16`
- isComplexType: `c10::complex<c10::Half>, c10::complex<float>, c10::complex<double>`

```
isComplexType(common_dtype)
AT_DISPATCH_FLOATING_TYPES_AND2(at::ScalarType::Half, at::ScalarType::BFloat16
```

rsqrt,rsqrt_,rsqrt.out

Pytorch 文档

- https://pytorch.org/docs/stable/generated/torch.rsqrt.html

Pytorch 实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/UnaryOpsKernel.cu#L96
- 同样是 gpu_kernel + elementwise functor

Unary op， 接口定义同 neg

代码块

```
Tensor rsqrt(const Tensor & self); // {"schema": "aten::rsqrt(Tensor self) -> Tensor", "dispatch": "True", "default": "True"}
Tensor & rsqrt_(Tensor & self); // {"schema": "aten::rsqrt_(Tensor(a!) self) -> Tensor(a!)", "dispatch": "True", "default": "True"}
Tensor & rsqrt_out(const Tensor & self, Tensor & out); // {"schema": "aten::rsqrt.out(Tensor self, *, Tensor(a!) out) -> Tensor(a!)", "dispatch": "True", "default": "False"}
```

Dtypes

- AT_FLOATING_TYPES: `float, double`
- Half, BFloat16: `float16, bfloat16`
- isComplexType: `c10::complex<c10::Half>, c10::complex<float>, c10::complex<double>`

代码块

```
AT_DISPATCH_FLOATING_TYPES_AND2(
      ScalarType::BFloat16, ScalarType::Half,
inline bool isComplexType(ScalarType t) {
  return (
      t == ScalarType::ComplexHalf || t == ScalarType::ComplexFloat ||
      t == ScalarType::ComplexDouble);
}
```

首个版本开发说明

（1）数据类型支持float32，bfloat16，float16

代码块

```
//calculate the reciprocal of the square-root of the input tensor with size dim0*dim1.
//total size should be 1024B algined
template <typename T>
//only sifmt::bfloat16, sifmt::float16, //sifmt::float32 are supported
void rsqrt(void *d_in, void *d_out, int dim0_size, int dim1_size );
```

sort,sort.stable,sort.values_stable

Pytorch 文档

- https://pytorch.org/docs/stable/generated/torch.sort.html

Pytorch 实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/SortStable.cu#L218
- 基于基数排序实现，cub 相关 apis 具体实现在 https://github.com/NVIDIA/cccl/blob/main/cub/cub/device/

代码块

```
// 在 RegisterCUDA.cpp 实现了 sort/sort_out 接口，调用的同样是 structured_sort_stable_out
::std::tuple<Tensor,Tensor> sort(const Tensor & self, ::std::optional<bool> stable, int64_t dim, bool descending); // {"schema": "aten::sort.stable(Tensor self, *, bool? stable, int dim=-1, bool descending=False) -> (Tensor values, Tensor indices)", "dispatch": "True", "default": "True"}
::std::tuple<Tensor &,Tensor &> sort_out(const Tensor & self, ::std::optional<bool> stable, int64_t dim, bool descending, Tensor & values, Tensor & indices); // {"schema": "aten::sort.values_stable(Tensor self, *, bool? stable, int dim=-1, bool descending=False, Tensor(a!) values, Tensor(b!) indices) -> (Tensor(a!) values, Tensor(b!) indices)", "dispatch": "True", "default": "False"}
```

Dtypes

- AT_ALL_TYPES: `uint8_t, int8_t, int16_t, int, int64_t, float, double`
- Half, Bool, BFloat16: `float16, bool, bfloat16`

代码块

```
AT_DISPATCH_ALL_TYPES_AND3(
        kBool, kHalf, kBFloat16
```

reciprocal,reciprocal_,reciprocal.out

Pytorch 文档

- https://pytorch.org/docs/stable/generated/torch.reciprocal.html
- 输入为 integral types 时，会将 input 转换为 float types，具体的处理逻辑在 TensorIteratror 的 UNARY_FLOAT_OP_CONFIG（） 里的 promote_inputs_to_common_dtype 以及 compute_types 里。因此需要处理的 dtypes 为 float/complex types

Pytorch 实现

- https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/UnaryFractionKernels.cu#L99
- 实现为 gpu_kernel + unary functor

Unary op，接口定义同 neg

代码块

```
Tensor reciprocal(const Tensor & self); // {"schema": "aten::reciprocal(Tensor self) -> Tensor", "dispatch": "True", "default": "True"}
Tensor & reciprocal_(Tensor & self); // {"schema": "aten::reciprocal_(Tensor(a!) self) -> Tensor(a!)", "dispatch": "True", "default": "True"}
Tensor & reciprocal_out(const Tensor & self, Tensor & out); // {"schema": "aten::reciprocal.out(Tensor self, *, Tensor(a!) out) -> Tensor(a!)", "dispatch": "True", "default": "False"}
```

dtypes

- AT_FLOATING_TYPES: `float, double`
- AT_COMPLEX_TYPES: `c10::complex<float>, c10::complex<double>`
- Half, BFloat16: `float16, bfloat16`

代码块

```
AT_DISPATCH_FLOATING_AND_COMPLEX_TYPES_AND2(
        ScalarType::Half, ScalarType::BFloat16
```

where.self,where.self_out,where,where.ScalarSelf,where.ScalarOther,where.Scalar

具体选择哪个搜索 static PyObject * THPVariable_where（PyObject* self_， PyObject* args，PyObject* kwargs）

参考信息

参考实现：

https://github.com/pytorch/pytorch/blob/9cd52da45c3030601bdd27e8a948f23be12b21ee/aten/src/ATen/native/TensorCompare.cpp#L631

资料：

https://pytorch.org/docs/stable/generated/torch.where.html

其中，where.self 通过 where.self_out 实现，可以搜索 Tensor& where_self_out（const Tensor& condition， const Tensor& self， const Tensor& other， Tensor& out）

Function 接口

~~self 和 other 的 tensor size 和 output tensor 一致~~

说明：condition判断条件，pytorch会把condition处理为一个bool tensor， 该tensor的shape和out完全一致。该算子实现，对于out[i], 根据condition tensor，如果该位置为true，就赋值self[i], 否则赋值other[i]。不区分scalar还是tensor，框架侧padding广播好数据。

代码块

```
// schema: where.self_out(Tensor condition, Tensor self, Tensor other, *, Tensor(a!) out) -> Tensor(a!)

/**
out_ndims: output tensor dims
out_sizes: output tensor sizes
out_strides: strides of output tensor, 0 means it's a broadcasted dim
self_strides: strides of self tensor, 0 means it's a broadcasted dim
other_strides: strides of self tensor, 0 means it's a broadcasted dim
element_sizes: element_size of nargs tensor; if element_sizes is nullptr, then the strides will be in bytes, otherwise the strides will be in # of elements.
 **/
~~void fn(dev, const void* out, const void* self, const void* other, int out_ndims, const int64_t* out_sizes, const int64_t* out_strides, const int64_t* self_strides, const int64_t* other_strides, const int64_t* element_sizes=nullptr)~~

接口修改如下：
template<typename T>
void where(const void* out, const void* condition, const void* self, const void* other, int out_ndims, const int64_t* out_shapes, const int64_t* out_strides, const int64_t* self_strides, const int64_t* other_strides)
```

```
16
```

数据类型

- kComplexHalf, kHalf, kBFloat16, kBool
- AT_EXPAND(AT_ALL_TYPES_AND_COMPLEX)：c10::kByte, c10::kChar, c10::kInt, c10::kLong, c10::kShort，c10::kDouble, c10::kFloat
- AT_EXPAND(AT_FLOAT8_TYPES))

代码例子

代码块

```
import torch

x = torch.randn(3, 2)
print(x)
y = torch.where(x > 0, -1, torch.tensor(1))
print(y)

####输出结果######
x=tensor([[-0.9950, -0.7007],
        [ 0.1992, -0.4350],
        [ 0.6579, -0.3728]])
self_converted.sizes: [], self_converted.strides: []  # scalar tensor的sizes和strides为空
condition tensor:  0  0
 1  0
 1  0
[ CPUBoolType{3,2} ]
out shapes: 6,   #  iter.shape()
iter tensor1 strides: [2, 1], iter tensor1 sizes: [3, 2]
iter tensor2 strides: [], iter tensor2 sizes: []   # scalar tensor经过tensoriterator处理，size和strides仍为空
iter tensor3 strides: [], iter tensor3 sizes: []
iter strides(1): [1]   # 广播后会把condition展平为一维处理，压缩为线性。coalesce_dimensions处理逻辑。
iter strides(2): [0]
iter strides(3): [0]
y=tensor([[ 1,  1],
        [-1,  1],
        [-1,  1]])
```

_to_copy

注册在 build/aten/src/ATen/RegisterCompositeExplicitAutograd.cpp

使用到的 op：zeros_like， empty_strided， copy_，empty_symint

view, view_as, reshape, _reshape_alias

参考资料：https://pytorch.org/docs/stable/generated/torch.Tensor.view.html

https://pytorch.org/docs/stable/generated/torch.reshape.html

https://pytorch.org/docs/stable/generated/torch.Tensor.view_as.html

view， view_as 和 reshape 底层调用的都是 **alias_with_sizes_and_strides 中的 make_tensor，不需要算子实现**

代码块

```
template <template <typename...> typename Container>
Tensor alias_with_sizes_and_strides(
    const Tensor& self,
    const Container<c10::SymInt>& sizes,
    const Container<c10::SymInt>& strides) {
  //caller should make sure that sizes and strides are valid for self
  //(storage is sufficient, strides are non-negative, strides and sizes array size is the same)
  Tensor self_;
  if (self.is_quantized()) {
    self_ = at::detail::make_tensor<QTensorImpl>(
      c10::TensorImpl::VIEW, Storage(self.storage()), self.key_set(), self.dtype(), get_qtensorimpl(self)->quantizer());
    self_.unsafeGetTensorImpl()->set_sizes_and_strides(sizes, strides, self.sym_storage_offset());
  } else {
    self_ = at::detail::make_tensor<TensorImpl>(
    c10::TensorImpl::VIEW, Storage(self.storage()), self.key_set(), self.dtype());
    self_.unsafeGetTensorImpl()->set_sizes_and_strides(sizes, strides, self.sym_storage_offset());
  }
  namedinference::propagate_names(self_, self);
  return self_;
}
```

view 操作要求能够找到一组有效的步长，使得在不重新排列内存的情况下可以实现形状变换。这在以下情况下可能失败：

1. 非连续张量：如果张量经过了转置等操作，导致内存布局不连续
2. 跨越不连续子空间的维度：当新形状要求将不连续的内存区域视为连续时

例如，对于一个形状为 [2， 3， 4] 的张量，如果经过 permute（2， 0， 1） 变为 [4， 2， 3]，然后尝试 view（8， 3），这将失败，因为无法找到合适的步长。

与 view 一样，view_as 也要求张量在内存中是连续的

reshape：内部会先检查是否可以直接使用 view；如果不行，会先调用 contiguous（） 创建一个连续的副本，再对这个副本使用 view

代码块

```
x = torch.randn(2, 3, 4)

# 将张量展平为一维
flat_view = x.view(-1)
print("展平后:", flat_view.shape)  # torch.Size([24])

# 改变为 2x12 的形状，第二维自动计算
reshaped = x.reshape(2, -1)
print("自动推断维度:", reshaped.shape)  # torch.Size([2, 12])


# 添加一个维度（批次维度）
batched = x.view(1, 2, 3, 4)
print("添加批次维度:", batched.shape)  # torch.Size([1, 2, 3, 4])

# 移除一个维度（当该维度大小为1时）
squeezed = batched.view(2, 3, 4)
print("移除批次维度:", squeezed.shape)  # torch.Size([2, 3, 4])


# 创建一个非连续张量（通过转置）
y = torch.randn(2, 3, 4)
y_transposed = y.transpose(0, 2)  # 形状变为 [4, 3, 2]
print("转置后是否连续:", y_transposed.is_contiguous())  # False

# 使用 view - 会失败
try:
    y_view = y_transposed.view(4, 6)
except RuntimeError as e:
    print("view 错误:", e)

# 使用 reshape - 会成功
y_reshape = y_transposed.reshape(4, 6)
print("reshape 成功:", y_reshape.shape)  # torch.Size([4, 6])

# 先使用 contiguous() 再使用 view
```

```python
y_view = y_transposed.contiguous().view(4, 6)
print("contiguous().view() 成功:", y_view.shape)  # torch.Size([4, 6])


x = torch.randn(4, 6)  # 形状为 [4, 6]
y = torch.randn(2, 12)  # 形状为 [2, 12]

# 使用 view_as 将 x 重塑为与 y 相同的形状
x_reshaped = x.view_as(y)

print("原始 x 形状:", x.shape)
print("y 形状:", y.shape)
print("重塑后的 x 形状:", x_reshaped.shape)
# 原始 x 形状: torch.Size([4, 6])
# y 形状: torch.Size([2, 12])
# 重塑后的 x 形状: torch.Size([2, 12])
```

uniform_,uniform,uniform.out

参考信息

Pytorch 文档

- https://pytorch.org/docs/stable/generated/torch.Tensor.uniform_.html

Pytorch 实现

- 调用了 curand 相关 api，generator impl 会调 用到 cuda graph 相关 api
- Kernel 实现：https://github.com/pytorch/pytorch/blob/v2.5.0/aten/src/ATen/native/cuda/DistributionTemplates.h#L66
- api 会需要支持传一个 generator，kernel 实现里会调用 generator 去具体生成，建议等支持第二阶段联合编译后再实现该 kernel

代码块

```
Tensor & uniform_(Tensor & self, double from, double to,
::std::optional<Generator> generator); // {"schema":
"aten::uniform_(Tensor(a!) self, float from=0, float to=1, *, Generator?
generator=None) -> Tensor(a!)", "dispatch": "True", "default": "False"}
```

代码块

```
x = torch.empty(2, 3)
x.uniform_(from_=0, to=1)
x
```

```
tensor([[0.2154, 0.5179, 0.8284],
        [0.4732, 0.9251, 0.3088]])
g = torch.Generator().manual_seed(42)

y = torch.empty(2, 3)
y.uniform_(from_=10, to=20, generator=g)
y
tensor([[16.6065, 18.1294, 17.8853],
        [18.9614, 13.6132, 14.2703]])
```

ISSUE

sym_size.int

把 **符号形状（symbolic size）** 转换成实际的整数（int64_t）值。

代码块

```
x = torch.zeros(2, 3, 4)
sym_dims = x.sym_size()
sym_dims
[2, 3, 4]
```

参考信息

Pytorch 实现

https://github.com/pytorch/pytorch/blob/9a0f7a3bb01b235ea04581ee540970a098071b72/aten/src/ATen/core/TensorBase.h#L165

```
243       c10::SymIntArrayRef sym_sizes() const {
244         return impl_->sym_sizes();
```

https://github.com/pytorch/pytorch/blob/9a0f7a3bb01b235ea04581ee540970a098071b72/aten/src/ATen/core/TensorBase.h#L1053

```
1053    template <typename T, typename = enable_if_symint<T>>
1054    c10::SymIntArrayRef sizes(const TensorBase& t) { return t.sym_sizes(); }
```

https://github.com/pytorch/pytorch/blob/9a0f7a3bb01b235ea04581ee540970a098071b72/aten/src/ATen/core/TensorBase.h#L243

```
165       c10::SymInt sym_size(int64_t dim) const {
166         return impl_->sym_size(dim);
167       }
```

https://github.com/pytorch/pytorch/blob/9a0f7a3bb01b235ea04581ee540970a098071b72/aten/src/ATen/core/TensorBase.h#L1058

```
1058    template <typename T, typename = enable_if_symint<T>>
1059    c10::SymInt size(const TensorBase& t, int64_t dim) { return t.sym_size(dim); }
```

【NOTE】不需要实现，功能由pytorch内部完成

polar，polar.out

参考信息

pytorch文档：https://pytorch.org/docs/stable/generated/torch.polar.html

参考实现：
https://github.com/pytorch/pytorch/blob/183bca41de89825107dce1d1ecc1502d9993a684/aten/src/ATen/native/cuda/ComplexKernel.cu#L22

代码块

```
>>> abs = torch.tensor([1.0, 2.0, 3.0])
>>> angle = torch.tensor([0.0, torch.pi/2, torch.pi])
>>> complex_tensor = torch.polar(abs, angle)
>>> print(complex_tensor)
tensor([ 1.0000e+00+0.0000e+00j, -8.7423e-08+2.0000e+00j,
        -3.0000e+00-2.6227e-07j])
```

Function 接口

代码块

```
/**
 * @brief Converts polar coordinates (magnitude, angle) to complex numbers
 *
 * Performs element-wise conversion from polar coordinates (r, θ) to complex numbers
 * using the formula: z = r * (cos(θ) + i*sin(θ))
 *
 * @param dev              Device handle for CUDA device management
 * @param out              Output complex tensor to store results
 * @param abs              Input tensor containing magnitude values (r)
 * @param angle            Input tensor containing angle values (θ) in radians
 * @param out_sizes        Output tensor sizes
 * @param out_strides      Output tensor strides
 * @param abs_sizes        abs tensor sizes
 * @param abs_strides      abs tensor strides
 * @param angle_sizes      Angle tensor sizes
```

```
16    * @param angle_strides    Angle tensor strides
17    * @param ndims            Number of dimensions in tensors
18    * @param out_dtype        Output complex data type (complex32/64/128)
19    * @param input_dtype      Input floating point data type (float16/32/64)
20    *
21    * @note Supported Data Types:
22    *       Input (magnitude, angle):
23    *       - float16 (half)
24    *       - bfloat16
25    *       - float32 (float)
26    *       - float64 (double)
27    *
28    *       Output (complex):
29    *       - complex32 (half complex)
30    *       - complex64 (single complex)
31    *       - complex128 (double complex)
32    *
33    * @note Memory Layout:
34    *       - Supports non-contiguous tensors
35    *       - Handles arbitrary strides efficiently
36    *       - Input tensors must have matching sizes
37    *       - Output complex tensor has same shape as inputs
38    *
39    * @note Implementation Details:
40    *       - Uses fast math functions for trigonometric operations
41    *       - Vectorized operations where possible
42    *       - Handles edge cases (inf, nan) according to IEEE standards
43    *       - Thread-safe and optimized for GPU execution
44    */
45   void polar_kernel_cuda(
46       SiDeviceHandle *dev,           // Device handle
47       void *out,                     // Output complex tensor
48       const void *abs,               // Input magnitude tensor
49       const void *angle,             // Input angle tensor
50       const int64_t* out_sizes,      // Output sizes
51       const int64_t* out_strides,    // Output strides
52       const int64_t* abs_sizes,      // Magnitude sizes
53       const int64_t* abs_strides,    // Magnitude strides
54       const int64_t* angle_sizes,    // Angle sizes
55       const int64_t* angle_strides,  // Angle strides
56       int ndims,                     // Number of dimensions
57       int out_dtype,                 // Output complex dtype
58       int input_dtype                // Input float dtype
59   );
```

ISSUE

- [ ] 使用TensorIterator 传递参数，支持非contiguous tensor

rms_norm

参考信息

参考实现：
https://github.com/pytorch/pytorch/blob/a05cc9f4941bd3b93e44b21cb7dac47d37003b32/aten/src/ATen/native/layer_norm.cpp#L264

Pytorch资料：
https://pytorch.org/docs/stable/generated/torch.nn.modules.normalization.RMSNorm.html

代码块

```
func: rms_norm(Tensor input, int[] normalized_shape, Tensor? weight=None, float? eps=None) -> Tensor

##基本用法
import torch
import torch.nn.functional as F

input = torch.randn(2, 3, 4, 5)
normalized_shape = [4, 5]  # 最后两个维度
weight = torch.ones(4, 5)  # 可选的缩放参数

output = torch.nn.functional.rms_norm(
    input,
    normalized_shape,
    weight=weight,  # 可选
    eps=1e-6        # 可选
)
```

Function 接口

代码块

```
/**
 * @brief Performs Root Mean Square (RMS) Normalization on input tensor
 *
 * RMS Normalization normalizes the input tensor by dividing it by the root mean square,
 * computed over the specified dimensions. It differs from Layer Normalization by not
 * centering the inputs (no mean subtraction), which makes it more computationally efficient.
 *
```

```
 * Formula: y = [x / sqrt(mean(x^2) + eps)] * weight (if provided)
 *
 * @param out              Rms_norm result
 * @param input            Input tensor to be normalized
 * @param weight           Optional weight tensor for scaling the normalized output
 *                         Must be of shape normalized_shape if provided
 * @param eps              Small constant added to denominator for numerical stability
 *                         If not provided, uses default epsilon for the data type
 * @param batch_size       numel of input / numel of normalized tensor
 * @param normalized_size  padded numel of normalize tensor
 *                         Normalization will be performed over the last dims of size normalized_shape
 * @param original_normalized_size  the original size of normalized_size
 * @param ndims            input/output ndims
 * @param normalized_shape_ndims   how many dims will be normalized
 * @param weight_opt       weither has weight input
 * @param eps_opt          weither has eps input
 *
 */
 void rms_norm_kernel(
    SiDeviceHandle *dev,              // Device handle
    void *output,                        // Output
    const void *input,                // Input tensor
    const void *weight,               // Optional weight tensor for scaling the normalized output
    const float eps,                  // Small constant added to denominator for numerical stability If not provided, uses default epsilon for the data type
    const int64_t batch_size,         // numel of input / numel of normalized tensor 为非normalize维度的乘积
    const int64_t normalized_size,    // padded numel of normalize tensor, 按照512pad之后的original_normalized_size
    const int64_t original_normalized_size, //original numel of normalize tensor 一般为input的最后几个维度的乘积
    int weight_opt,                   // 0 for false, 1 for true
    int eps_opt,                      // 0 for false, 1 for true
);

rms_norm的input size = output size

weight和eps都是可选参数，其中weight的shape是normalized_shape，eps Defaults to 1e-6

传入input shape=[batch_size, normalized_size]
weight shape = [normalized_size]
out shape = [batch_size, normalized_size]
```

```
input cuda支持的输入类型
1. float
2. double
3. complex<float>
4. complex<double>
5. Half (float16)
6. BFloat16
```

ISSUE

- [ ] 虽然cuda的实现中可以接收非contiguous的input，但是对于contiguous的支持是非必要的，可以在pytorch侧先将输入转换为contiguous。

amax, amax.out

参考信息

pytorch文档：https://pytorch.org/docs/stable/generated/torch.amax.html

参考实现：和max共用一个kernel

与max的不同：

1. **维度reduce方面**：amax/amin能够支持在多个维度上进行**reduce**操作，而max/min在这一点上不具备这样的功能。
2. **是否返回索引**：amax/amin在执行操作时不会返回对应的索引，而max/min则会返回索引。
3. **梯度传播方式**：当存在多个相等的值时，amax/amin会将梯度均匀地分配到这些相等的值上，而max(dim)/min(dim)在传播梯度时，只会将梯度传播到源张量中的一个索引位置。

使用方法：

代码块

```
>>> x = torch.tensor([[1, 2, 3], [4, 5, 6]])
>>> max_val = torch.amax(x)
>>> max_val
tensor(6)

>>> x = torch.tensor([[1, 2, 3], [4, 5, 6]])
>>> max_dim0 = torch.amax(x, dim=0)
>>> max_dim0
tensor([4, 5, 6])

>>> x = torch.tensor([[1, 2, 3], [4, 5, 6]])
>>> max_dim1_keep = torch.amax(x, dim=1, keepdim=True)
```

```
>>> max_dim1_keep
tensor([[3],
        [6]])

>>> x = torch.randn(2, 3, 4)
>>> x
tensor([[[ 1.3958e+00,  1.0015e+00, -3.7019e-01, -1.5914e+00],
         [-7.9525e-01,  3.4663e-02, -1.8195e+00, -1.3115e-03],
         [ 5.5305e-01,  2.5448e+00,  7.1271e-01,  9.1277e-02]],

        [[ 1.0165e+00,  2.1791e-01,  1.3918e-01, -1.5510e-01],
         [-8.1696e-01,  3.8472e-01,  1.5919e+00,  4.4249e-01],
         [-5.2152e-01,  6.0341e-01,  9.4258e-02,  2.8002e-01]]])
>>> max_multi = torch.amax(x, dim=(0, 1))
>>> max_multi
tensor([1.3958, 2.5448, 1.5919, 0.4425])
```

Function 接口

参考：OP 的开发文档

与max不同的Reduce dim的处理和output处理在sipu torch侧处理

bincount,bincount.out

参考信息

pytorch文档：https://pytorch.org/docs/2.6/generated/torch.bincount.html#torch-bincount

cuda参考实现：
https://github.com/pytorch/pytorch/blob/1a6d50d40799ba05b17254b236a6bf8817dd7d06/aten/src/ATen/native/cuda/SummaryOps.cu#L248

注意：只能处理一维的非负整数张量

代码块

```
>>> x = torch.tensor([1, 2, 2, 3, 3, 3])
>>> counts = torch.bincount(x)
>>> counts
tensor([0, 1, 2, 3])

>>> x = torch.tensor([1, 1, 2])
>>> counts = torch.bincount(x, minlength=5)
>>> counts
tensor([0, 2, 1, 0, 0])

```

```
>>> x = torch.tensor([1, 1, 2, 3])
>>> weights = torch.tensor([0.1, 0.2, 0.3, 0.4])
>>> weighted_counts = torch.bincount(x, weights=weights)
>>> print(weighted_counts)
tensor([0.0000, 0.3000, 0.3000, 0.4000])
```

Function 接口

代码块

```
/**
 * @brief Counts the frequency of each value in a non-negative integer tensor
 *
 * Bincount computes the occurrence count of each value in the input tensor.
 * It can optionally apply weights to compute weighted sums instead of counts.
 *
 * Formula:
 *   - Without weights: out[i] = count of i in input
 *   - With weights: out[i] = sum(weights[j] where input[j] == i)
 *
 * @param output           Output tensor containing the counts/sums
 * @param self             Input tensor (must be 1D non-negative integers)
 * @param weights          Optional weights tensor for weighted sums
 *                         Must be same size as input if provided
 * @param minlength        Minimum length of output tensor
 * @param input_size       Number of elements in input tensor
 * @param output_size      Size of output tensor (max(input) + 1 or minlength)
 * @param has_weights      Flag indicating if weights are provided (0=false,
1=true)
 */
void bincount_kernel(
    SiDeviceHandle* dev,          // Device handle
    void* output,                 // Output tensor buffer
    const void* self,             // Input tensor data
    const void* weights,          // Optional weights data
    int64_t minlength,            // Minimum output length
    int64_t input_size,           // Number of elements in input
    int64_t output_size,          // Size of output tensor
    int has_weights               // Whether weights are provided
);

self支持的数据类型:
uint8_t (通过特例检查)
int8_t
int16_t
int32_t
int64_t
```

```
37
38  weights支持的数据类型：
39  float
40  double
```

首个版本开发说明

（1）~~数制支持上，初版需要支持int64_t类型~~

（2）初版暂时不用支持weights参数

（3）输出tensor的长度上，初版先依赖min len，后续考虑是否支持max + 1的情况

（4）新情况变更，由于硬件限制，初版支持int32_t类型，由框架将数据进行64-32转换

内部接口调用形式：

代码块

```
1  template <typename T>
2  void bincount(void *d_in, void *d_out, T dim_size, T min_len);
```

ISSUE

- [ ] 输入必须是1维连续的Strided布局张量,不支持任何形式的非连续或特殊布局
- [ ] 只能处理一维的非负整数张量

nonzero,nonzero.out

参考信息

pytorch文档：https://pytorch.org/docs/stable/generated/torch.nonzero.html

参考实现：

https://github.com/pytorch/pytorch/blob/a936d596f6f7d2bc2dc47b4b2320208b4908e7f2/aten/src/ATen/native/cuda/Nonzero.cu#L170

代码块

```
1  >>> x = torch.tensor([[1, 0, 0], [0, 2, 0], [0, 0, 3]])
2  >>> indices = torch.nonzero(x)
3  >>> indices
4  tensor([[0, 0],
5          [1, 1],
6          [2, 2]])
7
8  >>> x = torch.tensor([[1, 0, 0], [0, 2, 0], [0, 0, 3]])
```

```
>>> rows, cols = torch.nonzero(x, as_tuple=True)
>>> rows
tensor([0, 1, 2])
>>> cols
tensor([0, 1, 2])

>>> x = torch.tensor([[[1, 0], [0, 2]], [[0, 0], [3, 0]]])
>>> indices = torch.nonzero(x)
>>> indices
tensor([[0, 0, 0],
        [0, 1, 1],
        [1, 1, 0]])

mask = torch.tensor([[True, False], [False, True]])
indices = torch.nonzero(mask)
print(indices)
# 输出:
# tensor([[0, 0],
#         [1, 1]])

x = torch.tensor([1, 2, 3, 4, 5])
indices = torch.nonzero(x > 3)
print(indices)  # tensor([[3], [4]])

>>> x = torch.tensor([[[0, 0], [0, 0]], [[0, 0], [0, 0]]])
>>> indices = torch.nonzero(x)
>>> indices
tensor([], size=(0, 3), dtype=torch.int64)
```

在cuda实现中，该算子是动态的，需要分两次调用cuda算子，

1. 第一次计算self中包含多少非0元素num_nonzeros_h,
   a. cub::TransformInputIterator<bool, NonZeroOp<scalar_t>, const scalar_t*> itr(self_.const_data_ptr<scalar_t>(), NonZeroOp<scalar_t>());将input全部转为0/1，然后通过sum来计算出来非0元素
2. 根据第一次调用返回的数据，初始化output的size为{num_nonzeros，ndim}
3. 第二次调用cuda，得到indicates，如果使用tuple，需要在pytorch侧进行后续处理，不在kernel中处理

Function 接口

这里设计的是第二次调用cuda使用的接口，第一次调用可参考 OP 的开发文档 sum 接口

```
代码块
```

```
/**
 * @brief Finds indices of non-zero elements in a tensor
 *
 * Nonzero computes the indices of all non-zero elements in the input tensor.
 * The output is a 2D tensor where each row contains the indices of a non-zero element.
 *
 * @param out              Output tensor containing the indices
 * @param self             Input tensor
 * @param self_sizes       input shape
 * @param self_strides     input strides
 * @param num_nonzeros     Number of non-zero elements in input
 * @param ndim             Number of dimensions of input tensor
 */
void nonzero_kernel(
    SiDeviceHandle* dev,          // Device handle
    void* out,                    // Output tensor buffer
    const void* self,             // Input tensor data
    const int64_t* self_sizes,    // input shape
    int64_t num_nonzeros,         // Number of non-zero elements
    int64_t ndim,                 // Number of dimensions
);

支持的数据类型
float32
float64
int8/uint8
int16/int32/int64
bool
bfloat16
complex64/complex128
```

首个版本开发说明

（1）数制支持上，需要支持int32_t，bool

（2）支持as_tuple参数

as_tuple为true时，返回tensor为d个，分别存储不同维度的坐标值（二维输入即存储坐标的x值，y值）

as_tuple为false时，返回tensor为1个，shape为[MAX_NUM, DIM_NUM]

（3）由于硬件限制，对于int64_t类型，由框架将数据进行64-32转换

（4）初版仅实现二维输入接口，输出也为二维，按照最大值配置空间，假设input是[m, n]，输出每个tensor的大小为[m * n]，而每个输出tensor的真实大小，将存储于out_size返回参数中

as_tuple为true时，返回tensor为2个

as_tuple为false时，返回tensor为1个，shape为[m * n, 2]

内部接口调用形式：

代码块

```
/// @brief Returns 2 tensor containing the indices of all non-zero elements of input
/// @tparam T Datatype support [int32_t, bool]
/// @param[in] d_in Input tensor with shape [dim0_size, dim1_size]
/// @param[out] d_out0 Output tensor 0 with shape [*out_size]
/// @param[out] d_out1 Output tensor 1 with shape [*out_size]
/// @param[in] dim0_size Tensor shape param 0
/// @param[in] dim1_size Tensor shape param 1
/// @param[out] out_size Record nonzero element number, this param should be on DEVICE MEMORY
template <typename T>
void nonzero_2d(void *d_in, void *d_out0, void *d_out1, int dim0_size, int dim1_size, int *out_size);

/// @brief Returns 1 tensor containing the indices of all non-zero elements of input
/// @tparam T Datatype support [int32_t, bool]
/// @param[in] d_in Input tensor with shape [dim0_size, dim1_size]
/// @param[out] d_out Output tensor with shape [*out_size, 2]
/// @param[in] dim0_size Tensor shape param 0
/// @param[in] dim1_size Tensor shape param 1
/// @param[out] out_size Record nonzero element number, this param should be on DEVICE MEMORY
template <typename T>
void nonzero_2d_no_tuple(void *d_in, void *d_out, int dim0_size, int dim1_size, int *out_size);
```

ISSUE

- [ ] 输入到kernel之前，self会先做contiguous处理，kernel接受的均为contiguous数据

vllm_flash_attn_c::varlen_fwd，vllm_flash_attn_c::fwd_kvcache[vLLM]

参考信息

vLLM 调用的地方在：https://github.com/vllm-project/vllm/blob/e92694b6fe264a85371317295bca6643508034ef/vllm/v1/attention/backends/flash_attn.py#L194

在这里分为 FA2 和 FA3（hopper架构）：

https://github.com/vllm-project/flash-attention/blob/main/vllm_flash_attn/flash_attn_interface.py

flash-attention/vllm_flash_attn/flash_attn_interface.py at main · vllm-project/flash-attent···

```python
def flash_attn_varlen_func(
    q,
    k,
    v,
    max_seqlen_q,
    cu_seqlens_q,
    max_seqlen_k,
    cu_seqlens_k=None, # only used for non-paged prefill
    seqused_k=None,
    dropout_p=0.0,
    softmax_scale=None,
    causal=False,
    window_size: Optional[List[int]] = None,
    softcap=0.0, # 0.0 means deactivated
    alibi_slopes=None,
    deterministic=False,
    return_attn_probs=False,
    block_table=None,
    *,
    return_softmax_lse=False,
    out=None,
    fa_version: int = DEFAULT_FA_VERSION,
):
    """dropout_p should be set to 0.0 during evaluation
    Supports multi-query and grouped-query attention (MQA/GQA) by passing in K, V with fewer heads
    than Q. Note that the number of heads in Q must be divisible by the number of heads in KV.
    For example, if Q has 6 heads and K, V have 2 heads, head 0, 1, 2 of Q will attention to head
    0 of K, V, and head 3, 4, 5 of Q will attention to head 1 of K, V.

    If causal=True, the causal mask is aligned to the bottom right corner of the attention matrix.
    For example, if seqlen_q = 2 and seqlen_k = 5, the causal mask (1 = keep, 0 = masked out) is:
        1 1 1 1 0
        1 1 1 1 1
    If seqlen_q = 5 and seqlen_k = 2, the causal mask is:
        0 0
        0 0
        0 0
```

```
        1 0
        1 1
    If the row of the mask is all zero, the output will be zero.

    If window_size != (-1, -1), implements sliding window local attention. Query at position i
    will only attend to keys between
    [i + seqlen_k - seqlen_q - window_size[0], i + seqlen_k - seqlen_q + window_size[1]] inclusive.

    Arguments:
        q: (total_q, nheads, headdim), where total_q = total number of query tokens in the batch.
        k: (total_k, nheads_k, headdim), where total_k = total number of key tokens in the batch.
        v: (total_k, nheads_k, headdim), where total_k = total number of key tokens in the batch.
        cu_seqlens_q: (batch_size + 1,), dtype torch.int32. The cumulative sequence lengths
           of the sequences in the batch, used to index into q.
        cu_seqlens_k: (batch_size + 1,), dtype torch.int32. The cumulative sequence lengths
           of the sequences in the batch, used to index into kv.
        max_seqlen_q: int. Maximum query sequence length in the batch.
        max_seqlen_k: int. Maximum key sequence length in the batch.
        dropout_p: float. Dropout probability.
        softmax_scale: float. The scaling of QK^T before applying softmax.
            Default to 1 / sqrt(headdim).
        causal: bool. Whether to apply causal attention mask (e.g., for auto-regressive modeling).
        window_size: (left, right). If not (-1, -1), implements sliding window local attention.
        softcap: float. Anything > 0 activates softcapping attention.
        alibi_slopes: (nheads,) or (batch_size, nheads), fp32. A bias of
            (-alibi_slope * |i + seqlen_k - seqlen_q - j|)
            is added to the attention score of query i and key j.
        deterministic: bool. Whether to use the deterministic implementation of the backward pass,
            which is slightly slower and uses more memory. The forward pass is always deterministic.
        return_attn_probs: bool. Whether to return the attention probabilities. This option is for
           testing only. The returned probabilities are not guaranteed to be correct
           (they might not have the right scaling).
    Return:
        out: (total, nheads, headdim).
```

```
        softmax_lse [optional, if return_softmax_lse=True]: (nheads, total_q_seqlen). The
            logsumexp of each row of the matrix QK^T * scaling (e.g., log of the softmax
            normalization factor).
    """
    assert cu_seqlens_k is not None or seqused_k is not None, \
        "cu_seqlens_k or seqused_k must be provided"
    assert cu_seqlens_k is None or seqused_k is None, \
        "cu_seqlens_k and seqused_k cannot be provided at the same time"
    assert block_table is None or seqused_k is not None, \
        "seqused_k must be provided if block_table is provided"

    if softmax_scale is None:
        softmax_scale = q.shape[-1] ** (-0.5)
    # custom op does not support non-tuple input
    real_window_size: Tuple[int, int]
    if window_size is None:
        real_window_size = (-1, -1)
    else:
        assert len(window_size) == 2
        real_window_size = (window_size[0], window_size[1])
    q, k, v = [maybe_contiguous(x) for x in (q, k, v)]

    dummy_cu_seqlens_k = torch.empty_like(cu_seqlens_q)

    if fa_version == 2:
        out, softmax_lse = torch.ops._vllm_fa2_C.varlen_fwd(
            q, k, v,
            out,
            cu_seqlens_q,
            # cu_seqlens_k not used since we use seqused_k, but flash_api.cpp
            # still wants it so we pass all zeros
            dummy_cu_seqlens_k if cu_seqlens_k is None else cu_seqlens_k,
            seqused_k,
            None,
            block_table,
            alibi_slopes,
            max_seqlen_q,
            max_seqlen_k,
            dropout_p,
            softmax_scale,
            False,
            causal,
            real_window_size[0],
            real_window_size[1],
            softcap,
```

```
            return_softmax_lse and dropout_p > 0,
            None,
        )
    elif fa_version == 3:
        out, softmax_lse, _, _ = torch.ops._vllm_fa3_C.fwd(
            q, k, v,
            None, None,       # k_new, v_new
            out,
            cu_seqlens_q,
            cu_seqlens_k,     # cu_seqlens_k
            None,             # cu_seqlens_k_new
            None, seqused_k,  # seqused_q, seqused_k
            max_seqlen_q, max_seqlen_k,
            block_table,
            alibi_slopes,
            None,             # kv_batch_idx
            None, None,       # rotary_cos, rotary_sin
            None, None, None, # q_descale, k_descale, v_descale
            softmax_scale,
            causal,
            real_window_size[0], real_window_size[1],
            0,                # sink_token_length
            softcap,
            True,             # rotary_interleaved
            0,                # num_splits
            None,             # pack_gqa
            0,                # sm_margin
        )
    else:
        raise ValueError(f"Unsupported FA version: {fa_version}")
    return (out, softmax_lse) if return_softmax_lse else out


def flash_attn_with_kvcache(
    q,
    k_cache,
    v_cache,
    k=None,
    v=None,
    rotary_cos=None,
    rotary_sin=None,
    cache_seqlens: Optional[Union[(int, torch.Tensor)]] = None,
    cache_batch_idx: Optional[torch.Tensor] = None,
    cache_leftpad: Optional[torch.Tensor] = None,
    block_table: Optional[torch.Tensor] = None,
    softmax_scale=None,
```

```
    causal=False,
    window_size=(-1, -1),  # -1 means infinite context window
    softcap=0.0, # 0.0 means deactivated
    rotary_interleaved=True,
    alibi_slopes=None,
    num_splits=0,
    return_softmax_lse=False,
    *,
    out=None,
    # FA3 Only
    scheduler_metadata=None,
    q_descale=None,
    k_descale=None,
    v_descale=None,
    # Version selector
    fa_version: int = DEFAULT_FA_VERSION,
):
    """
    If k and v are not None, k_cache and v_cache will be updated *inplace* with the new values from
    k and v. This is useful for incremental decoding: you can pass in the cached keys/values from
    the previous step, and update them with the new keys/values from the current step, and do
    attention with the updated cache, all in 1 kernel.

    If you pass in k / v, you must make sure that the cache is large enough to hold the new values.
    For example, the KV cache could be pre-allocated with the max sequence length, and you can use
    cache_seqlens to keep track of the current sequence lengths of each sequence in the batch.

    Also apply rotary embedding if rotary_cos and rotary_sin are passed in. The key @k will be
    rotated by rotary_cos and rotary_sin at indices cache_seqlens, cache_seqlens + 1, etc.
    If causal or local (i.e., window_size != (-1, -1)), the query @q will be rotated by rotary_cos
    and rotary_sin at indices cache_seqlens, cache_seqlens + 1, etc.
    If not causal and not local, the query @q will be rotated by rotary_cos and rotary_sin at
    indices cache_seqlens only (i.e. we consider all tokens in @q to be at position cache_seqlens).

    See tests/test_flash_attn.py::test_flash_attn_kvcache for examples of how to use this function.
```

```
199
200     Supports multi-query and grouped-query attention (MQA/GQA) by passing in
    KV with fewer heads
201     than Q. Note that the number of heads in Q must be divisible by the number
    of heads in KV.
202     For example, if Q has 6 heads and K, V have 2 heads, head 0, 1, 2 of Q
    will attention to head
203     0 of K, V, and head 3, 4, 5 of Q will attention to head 1 of K, V.
204
205     If causal=True, the causal mask is aligned to the bottom right corner of
    the attention matrix.
206     For example, if seqlen_q = 2 and seqlen_k = 5, the causal mask (1 = keep,
    0 = masked out) is:
207         1 1 1 1 0
208         1 1 1 1 1
209     If seqlen_q = 5 and seqlen_k = 2, the causal mask is:
210         0 0
211         0 0
212         0 0
213         1 0
214         1 1
215     If the row of the mask is all zero, the output will be zero.
216
217     If window_size != (-1, -1), implements sliding window local attention.
    Query at position i
218     will only attend to keys between
219     [i + seqlen_k - seqlen_q - window_size[0], i + seqlen_k - seqlen_q +
    window_size[1]] inclusive.
220
221     Note: Does not support backward pass.
222
223     Arguments:
224         q: (batch_size, seqlen, nheads, headdim)
225         k_cache: (batch_size_cache, seqlen_cache, nheads_k, headdim) if
    there's no block_table,
226             or (num_blocks, page_block_size, nheads_k, headdim) if there's a
    block_table (i.e. paged KV cache)
227             page_block_size must be a multiple of 256.
228         v_cache: (batch_size_cache, seqlen_cache, nheads_k, headdim) if
    there's no block_table,
229             or (num_blocks, page_block_size, nheads_k, headdim) if there's a
    block_table (i.e. paged KV cache)
230         k [optional]: (batch_size, seqlen_new, nheads_k, headdim). If not
    None, we concatenate
231             k with k_cache, starting at the indices specified by cache_seqlens.
232         v [optional]: (batch_size, seqlen_new, nheads_k, headdim). Similar to
    k.
```

```
        rotary_cos [optional]: (seqlen_ro, rotary_dim / 2). If not None, we apply rotary embedding
            to k and q. Only applicable if k and v are passed in. rotary_dim must be divisible by 16.
        rotary_sin [optional]: (seqlen_ro, rotary_dim / 2). Similar to rotary_cos.
        cache_seqlens: int, or (batch_size,), dtype torch.int32. The sequence lengths of the
            KV cache.
        block_table [optional]: (batch_size, max_num_blocks_per_seq), dtype torch.int32.
        cache_batch_idx: (batch_size,), dtype torch.int32. The indices used to index into the KV cache.
            If None, we assume that the batch indices are [0, 1, 2, ..., batch_size - 1].
            If the indices are not distinct, and k and v are provided, the values updated in the cache
                might come from any of the duplicate indices.
        softmax_scale: float. The scaling of QK^T before applying softmax.
            Default to 1 / sqrt(headdim).
        causal: bool. Whether to apply causal attention mask (e.g., for auto-regressive modeling).
        window_size: (left, right). If not (-1, -1), implements sliding window local attention.
        softcap: float. Anything > 0 activates softcapping attention.
        rotary_interleaved: bool. Only applicable if rotary_cos and rotary_sin are passed in.
            If True, rotary embedding will combine dimensions 0 & 1, 2 & 3, etc. If False,
            rotary embedding will combine dimensions 0 & rotary_dim / 2, 1 & rotary_dim / 2 + 1
            (i.e. GPT-NeoX style).
        alibi_slopes: (nheads,) or (batch_size, nheads), fp32. A bias of
            (-alibi_slope * |i + seqlen_k - seqlen_q - j|)
            is added to the attention score of query i and key j.
        num_splits: int. If > 1, split the key/value into this many chunks along the sequence.
            If num_splits == 1, we don't split the key/value. If num_splits == 0, we use a heuristic
            to automatically determine the number of splits.
            Don't change this unless you know what you are doing.
        return_softmax_lse: bool. Whether to return the logsumexp of the attention scores.

    Return:
        out: (batch_size, seqlen, nheads, headdim).
```

```
263         softmax_lse [optional, if return_softmax_lse=True]: (batch_size,
    nheads, seqlen). The
264             logsumexp of each row of the matrix QK^T * scaling (e.g., log of
    the softmax
265             normalization factor).
266     """
267     assert k_cache.stride(-1) == 1, "k_cache must have contiguous last
    dimension"
268     assert v_cache.stride(-1) == 1, "v_cache must have contiguous last
    dimension"
269     q, k, v = [maybe_contiguous(x) for x in (q, k, v)]
270     if softmax_scale is None:
271         softmax_scale = q.shape[-1] ** (-0.5)
272     if cache_seqlens is not None and isinstance(cache_seqlens, int):
273         cache_seqlens = torch.full(
274             (k_cache.shape[0],), cache_seqlens, dtype=torch.int32,
    device=k_cache.device
275         )
276         cache_seqlens = maybe_contiguous(cache_seqlens)
277     cache_batch_idx = maybe_contiguous(cache_batch_idx)
278     block_table = maybe_contiguous(block_table)
279
280     if fa_version == 2:
281         if scheduler_metadata is not None and q_descale is not None \
282             and k_descale is not None and v_descale is not None:
283                 raise NotImplementedError(
284                     "FA2 does not support scheduler_metadata, q_descale, "
285                     "k_descale, v_descale"
286                 )
287         out, softmax_lse = torch.ops._vllm_fa2_C.fwd_kvcache(
288             q, k_cache, v_cache,
289             k, v,              # k_new, v_new
290             cache_seqlens,
291             rotary_cos,
292             rotary_sin,
293             cache_batch_idx,
294             cache_leftpad,
295             block_table,
296             alibi_slopes,
297             out,
298             softmax_scale,
299             causal,
300             window_size[0],
301             window_size[1],
302             softcap,
303             rotary_interleaved,
304             num_splits,
```

```
        )
    elif fa_version == 3:
        assert alibi_slopes is None, "Alibi is not supported in FA3"
        out, softmax_lse, _, _ = torch.ops._vllm_fa3_C.fwd(
            q, k_cache, v_cache, # q, k, v
            k, v,                # k_new, v_new
            None,                # q_v
            out,
            None, None,          # cu_seqlens_q, cu_seqlens_k
            None,                # cu_seqlens_k_new
            None, cache_seqlens, # seqused_q, seqused_k
            None, None,          # max_seqlen_q, max_seqlen_k
            block_table,
            cache_batch_idx,     # kv_batch_idx
            None,                # leftpad_k
            None, None,          # rotary_cos, rotary_sin
            q_descale, k_descale, v_descale,
            softmax_scale,
            causal,
            window_size[0], window_size[1],
            softcap,
            rotary_interleaved,  # rotary_interleaved
            scheduler_metadata,
            num_splits,          # num_splits
            None,                # pack_gqa
            0,                   # sm_margin
        )
    else:
        raise ValueError(f"Unsupported FA version: {fa_version}")
    return (out, softmax_lse) if return_softmax_lse else out
```

FA2 定义在：https://github.com/vllm-project/flash-attention/blob/d637d8927a35922ce6f6c0dff6dd3f765ed71f3c/flash_attn/flash_attn_interface.py#L1375

hopper 架构的 FA3 定义位置不同：https://github.com/vllm-project/flash-attention/blob/d637d8927a35922ce6f6c0dff6dd3f765ed71f3c/hopper/flash_attn_interface.py#L541

参考实现： 虽然使用的 kernel 不同，但是 FA2 和 FA3 的实现都是通过模板实例化

FA2: https://github.com/vllm-project/flash-attention/tree/main/csrc/flash_attn/src
flash_fwd_launch_template.h flash_fwd_kernel.h

FA3: https://github.com/vllm-project/flash-attention/blob/main/hopper/flash_fwd_launch_template.h https://github.com/vllm-project/flash-attention/blob/main/hopper/flash_api.cpp https://github.com/vllm-project/flash-attention/tree/main/hopper/instantiations

FA2接口可以参考

https://github.com/vllm-project/flash-attention/blob/main/csrc/flash_attn/flash_api.cpp

csrc/flash_attn/flash_api.cpp

```
PYBIND11_MODULE(TORCH_EXTENSION_NAME, m) {
    m.doc() = "FlashAttention";
    m.def("fwd", &FLASH_NAMESPACE::mha_fwd, "Forward pass");
    m.def("varlen_fwd", &FLASH_NAMESPACE::mha_varlen_fwd, "Forward pass (variable length)");
    m.def("bwd", &FLASH_NAMESPACE::mha_bwd, "Backward pass");
    m.def("varlen_bwd", &FLASH_NAMESPACE::mha_varlen_bwd, "Backward pass (variable length)");
    m.def("fwd_kvcache", &FLASH_NAMESPACE::mha_fwd_kvcache, "Forward pass, with KV-cache");
}
```

FA3接口可以参考

https://github.com/vllm-project/flash-attention/blob/main/hopper/flash_api.cpp

代码块

```
PYBIND11_MODULE(TORCH_EXTENSION_NAME, m) {
    m.doc() = "FlashAttention";
    m.def("fwd", &mha_fwd, "Forward pass");
    m.def("bwd", &mha_bwd, "Backward pass");
    m.def("fwd_combine", &mha_combine, "Combine partial attention outputs");
    m.def("get_scheduler_metadata", &mha_fwd_get_scheduler_metadata, "Get scheduler metadata for varlen forward pass");
}
```

FA的forward方法均是通过一个结构体Flash_fwd_params来传递参数，调用对应的kernel，但是FA2和FA3的Flash_fwd_params结构体定义是不同的

FA2 https://github.com/vllm-project/flash-attention/blob/d637d8927a35922ce6f6c0dff6dd3f765ed71f3c/csrc/flash_attn/src/flash.h#L48

FA3 https://github.com/vllm-project/flash-attention/blob/d637d8927a35922ce6f6c0dff6dd3f765ed71f3c/hopper/flash.h#L37

如果接口支持结构体传递，可以直接使用上面链接中的结构体

Function 接口

FA2

Without kvcache

代码块

```
/**
 * @brief Executes the Flash Attention forward pass with fully explicit parameters
 *
 * @param dev             Device handle for CUDA/CPU device management
 *                        Used to identify the device where tensors are allocated and computation will occur
 *
 * @param q_ptr           Pointer to query tensor data (total_q, nheads, headdim) or (batch_size, seqlen_q, nheads, headdim)
 *                        For variable length sequences, total_q = sum of all sequence lengths in batch
 *
 * @param k_ptr           Pointer to key tensor data (total_k, nheads_k, headdim) or (batch_size, seqlen_k, nheads_k, headdim)
 *                        For KV cache, this points to the cached keys
 *
 * @param v_ptr           Pointer to value tensor data (total_k, nheads_k, headdim) or (batch_size, seqlen_k, nheads_k, headdim)
 *                        For KV cache, this points to the cached values
 *
 * @param o_ptr           Pointer to output tensor data (total_q, nheads, headdim) or (batch_size, seqlen_q, nheads, headdim)
 *                        Will store the result of attention computation
 *
 * @param softmax_lse_ptr  Pointer to store log-sum-exp values from softmax computation (nheads, total_q) or (batch_size, nheads, seqlen_q)
 *                        Used for numerical stability and backward pass. Set to nullptr if not needed.
 *
 * @param p_ptr           Pointer to store attention matrix (batch_size, nheads, seqlen_q, seqlen_k)
 *                        Primarily used for debugging or visualization. Set to nullptr if not needed.
```

```
24   *
25   * @param batch_size       Number of sequences in the batch
26   *                         For fixed length: all sequences have the same length
27   *                         For variable length: sequences can have different
     lengths
28   *
29   * @param seqlen_q         Length of each query sequence (for fixed length) or
     maximum query sequence length (for variable length)
30   *                         In decoder scenarios, typically 1 for
     autoregressive generation
31   *
32   * @param seqlen_k         Length of each key/value sequence (for fixed
     length) or maximum key/value sequence length (for variable length)
33   *                         In KV cache scenarios, this is the current length
     of the cached context
34   *
35   * @param num_heads        Number of attention heads for query tensor
36   *                         Each head performs independent attention computation
37   *
38   * @param num_heads_k      Number of attention heads for key/value tensors
39   *                         For standard attention: equals num_heads
40   *                         For multi-query attention (MQA): equals 1
41   *                         For grouped-query attention (GQA): equals num_heads
     / group_size
42   *
43   * @param head_size        Dimension of each attention head
44   *                         Typically 64, 80, or 128 in modern architectures
45   *                         Must be ≤ 256 for Flash Attention implementation
46   *
47   * @param q_batch_stride   Stride (in elements) between consecutive batches in
     query tensor
48   *                         For contiguous tensor: seqlen_q * num_heads *
     head_size
49   *
50   * @param q_row_stride     Stride (in elements) between consecutive rows
     (sequences) in query tensor
51   *                         For contiguous tensor: num_heads * head_size
52   *
53   * @param q_head_stride    Stride (in elements) between consecutive heads in
     query tensor
54   *                         For contiguous tensor: head_size
55   *
56   * @param k_batch_stride   Stride (in elements) between consecutive batches in
     key tensor
57   *                         For contiguous tensor: seqlen_k * num_heads_k *
     head_size
58   *
```

```
59   * @param k_row_stride     Stride (in elements) between consecutive rows
     (sequences) in key tensor
60   *                         For contiguous tensor: num_heads_k * head_size
61   *
62   * @param k_head_stride    Stride (in elements) between consecutive heads in
     key tensor
63   *                         For contiguous tensor: head_size
64   *
65   * @param v_batch_stride   Stride (in elements) between consecutive batches in
     value tensor
66   *                         For contiguous tensor: seqlen_k * num_heads_k *
     head_size
67   *
68   * @param v_row_stride     Stride (in elements) between consecutive rows
     (sequences) in value tensor
69   *                         For contiguous tensor: num_heads_k * head_size
70   *
71   * @param v_head_stride    Stride (in elements) between consecutive heads in
     value tensor
72   *                         For contiguous tensor: head_size
73   *
74   * @param o_batch_stride   Stride (in elements) between consecutive batches in
     output tensor
75   *                         For contiguous tensor: seqlen_q * num_heads *
     head_size
76   *
77   * @param o_row_stride     Stride (in elements) between consecutive rows
     (sequences) in output tensor
78   *                         For contiguous tensor: num_heads * head_size
79   *
80   * @param o_head_stride    Stride (in elements) between consecutive heads in
     output tensor
81   *                         For contiguous tensor: head_size
82   *
83   * @param softmax_scale    Scale factor applied to attention scores before
     softmax
84   *                         Typically 1/sqrt(head_size) to normalize variance
85   *                         Affects the sharpness of attention distribution
86   *
87   * @param p_dropout        Dropout probability (0.0 to 1.0)
88   *                         Fraction of attention connections to randomly zero
     out during training
89   *                         Set to 0.0 for inference or to disable dropout
90   *
91   * @param softcap          Softcapping value for attention scores (0.0 to
     disable)
```

```
92   *                       If > 0.0, applies a soft cap to attention scores to
     prevent extreme values
93   *                       Helps with training stability for very long
     sequences
94   *
95   * @param is_causal        Whether to apply causal attention mask
96   *                         If true, each position can only attend to previous
     positions (including itself)
97   *                         Used in autoregressive models to prevent looking at
     future tokens
98   *
99   * @param window_size_left Left window size for local attention (-1 for
     unlimited)
100  *                         If >= 0, limits how far back each query can attend
101  *                         Useful for long sequence efficiency or implementing
     sliding window attention
102  *
103  * @param window_size_right Right window size for local attention (-1 for
     unlimited)
104  *                          If >= 0, limits how far forward each query can
     attend
105  *                          Only relevant when is_causal=false
106  *
107  * @param cu_seqlens_q     Cumulative sequence lengths for queries
     (batch_size+1)
108  *                         Format: [0, len1, len1+len2, len1+len2+len3, ...]
109  *                         Used for variable length sequences. Set to nullptr
     for fixed length.
110  *
111  * @param cu_seqlens_k     Cumulative sequence lengths for keys/values
     (batch_size+1)
112  *                         Format: [0, len1, len1+len2, len1+len2+len3, ...]
113  *                         Used for variable length sequences. Set to nullptr
     for fixed length.
114  *
115  * @param seqused_k        Actual sequence usage for keys (batch_size)
116  *                         Specifies how much of each sequence in the KV cache
     is actually used
117  *                         Useful when allocated length exceeds used length.
     Set to nullptr if not needed.
118  *
119  * @param leftpad_k        Left padding for each sequence in key (batch_size)
120  *                         Specifies number of padding tokens at the beginning
     of each sequence
121  *                         Affects position calculations for rotary
     embeddings. Set to nullptr if no left padding.
122  *
```

```
 * @param is_seqlens_k_cumulative Whether sequence lengths in cu_seqlens_k are cumulative
 *                                If true, cu_seqlens_k contains cumulative lengths
 *                                If false, cu_seqlens_k contains actual sequence lengths
 *
 * @param knew_ptr          New key tensor data pointer (batch_size, seqlen_knew, nheads_k, headdim)
 *                          Contains new keys to be appended to the KV cache
 *                          Set to nullptr if not appending to cache
 *
 * @param vnew_ptr          New value tensor data pointer (batch_size, seqlen_knew, nheads_k, headdim)
 *                          Contains new values to be appended to the KV cache
 *                          Set to nullptr if not appending to cache
 *
 * @param seqlen_knew       Length of new keys/values to append to the cache
 *                          Typically 1 for autoregressive generation, or chunk size for batch processing
 *
 * @param knew_batch_stride Stride between batches in new key tensor
 *                           For contiguous tensor: seqlen_knew * num_heads_k * head_size
 *
 * @param knew_row_stride   Stride between rows in new key tensor
 *                           For contiguous tensor: num_heads_k * head_size
 *
 * @param knew_head_stride  Stride between heads in new key tensor
 *                           For contiguous tensor: head_size
 *
 * @param vnew_batch_stride Stride between batches in new value tensor
 *                           For contiguous tensor: seqlen_knew * num_heads_k * head_size
 *
 * @param vnew_row_stride   Stride between rows in new value tensor
 *                           For contiguous tensor: num_heads_k * head_size
 *
 * @param vnew_head_stride  Stride between heads in new value tensor
 *                           For contiguous tensor: head_size
 *
 * @param rotary_cos_ptr    Cosine table for rotary embeddings (seqlen_ro, rotary_dim/2)
 *                           Contains precomputed cosine values for rotary position embeddings
 *                           seqlen_ro should be >= maximum sequence length. Set to nullptr to disable.
```

```
159  *
160  * @param rotary_sin_ptr    Sine table for rotary embeddings (seqlen_ro,
     rotary_dim/2)
161  *                          Contains precomputed sine values for rotary
     position embeddings
162  *                          seqlen_ro should be >= maximum sequence length.
     Set to nullptr to disable.
163  *
164  * @param rotary_dim        Dimension for rotary embeddings (must be <=
     head_size and divisible by 16)
165  *                          Only the first rotary_dim dimensions of each head
     will have rotary embeddings
166  *                          Set to 0 if not using rotary embeddings
167  *
168  * @param is_rotary_interleaved Whether rotary embeddings are interleaved
169  *                              If true, dimensions are paired as (0,1),
     (2,3), etc. (GPT-J style)
170  *                              If false, dimensions are paired as
     (0,rotary_dim/2), (1,rotary_dim/2+1), etc. (GPT-NeoX style)
171  *
172  * @param cache_batch_idx   Indices for batch mapping (batch_size)
173  *                          Maps from logical batch indices to physical
     indices in the KV cache
174  *                          Useful for reordering or sharing KV cache across
     requests. Set to nullptr for direct mapping.
175  *
176  * @param block_table       Block table for paged KV cache (batch_size,
     max_blocks_per_seq)
177  *                          Contains block indices for each sequence in paged
     KV cache
178  *                          Each row maps logical sequence positions to
     physical memory blocks. Set to nullptr for contiguous KV cache.
179  *
180  * @param block_table_batch_stride Stride between batches in block table
181  *                                 For contiguous tensor: max_blocks_per_seq
182  *
183  * @param page_block_size   Size of each block in paged KV cache
184  *                          Typically 16 or 32 tokens per block
185  *                          Only relevant when using paged KV cache
     (block_table != nullptr)
186  *
187  * @param alibi_slopes_ptr  ALiBi position bias slopes (nheads) or
     (batch_size, nheads)
188  *                          Contains slope values for ALiBi positional encoding
189  *                          Different slopes per head create different
     attention patterns. Set to nullptr to disable ALiBi.
190  *
```

```
191  * @param alibi_slopes_batch_stride Stride between batches in ALiBi slopes
192  *                                  For contiguous tensor: nheads
193  *                                  Set to 0 if alibi_slopes is shared across batch (shape is just (nheads))
194  *
195  * @param num_splits        Number of splits for computation (0 for auto)
196  *                          Splits the key/value tensors into chunks along sequence dimension
197  *                          Helps with memory efficiency for long sequences. 0 means auto-select based on GPU.
198  *
199  * @param oaccum_ptr        Accumulated output pointer for split computation (total_q, nheads, headdim)
200  *                          Used internally for split-KV optimization
201  *                          Set to nullptr unless manually implementing split computation
202  *
203  * @param softmax_lseaccum_ptr Accumulated LSE pointer for split computation (nheads, total_q)
204  *                             Used internally for split-KV optimization
205  *                             Set to nullptr unless manually implementing split computation
206  *
207  * @param is_bf16           Whether tensors are in BF16 format (true) or FP16 format (false)
208  *                          BF16 has less precision but more range than FP16
209  *                          Must accurately reflect the data type of input tensors
210  *
211  * @param unpadded_lse      Whether LSE is in unpadded format
212  *                          If true, softmax_lse has shape [nheads, total_q] for variable length sequences
213  *                          If false, softmax_lse has shape [batch_size, nheads, seqlen_q]
214  *
215  * @param seqlenq_ngroups_swapped Whether q has been transposed for optimization
216  *                                If true, q has been transposed from (b, 1, (nheads_k, ngroups), d) to (b, ngroups, nheads_k, d)
217  *                                Used for specialized optimizations in certain model architectures
218  *
219  * @param stream            CUDA stream to execute the kernel on
220  *                          Allows asynchronous execution and integration with other CUDA operations
221  *                          Set to nullptr to use the current CUDA stream
222  *
```

```
 * @param force_split_kernel Whether to force using the split kernel implementation
 *                           If true, always uses the split-KV optimization regardless of sequence length
 *                           Required for KV cache, paged KV cache, or when using cache_batch_idx
 *
 * @param return_softmax_lse Whether to return the log-sum-exp values
 *                           If true, softmax_lse_ptr must be a valid pointer
 *
 * @return Status code (0 for success, non-zero for failure)
 *         Specific error codes depend on the implementation
 */
int run_mha_fwd(
    void* dev,                      // Device handle
    // Core tensor pointers
    void* q_ptr,
    void* k_ptr,
    void* v_ptr,
    void* o_ptr,
    void* softmax_lse_ptr = nullptr,
    void* p_ptr = nullptr,

    // Dimensions
    int batch_size = 1,
    int seqlen_q = 0,
    int seqlen_k = 0,
    int num_heads = 0,
    int num_heads_k = 0,
    int head_size = 0,

    // Stride information (in elements, not bytes)
    int64_t q_batch_stride = 0,
    int64_t q_row_stride = 0,
    int64_t q_head_stride = 0,
    int64_t k_batch_stride = 0,
    int64_t k_row_stride = 0,
    int64_t k_head_stride = 0,
    int64_t v_batch_stride = 0,
    int64_t v_row_stride = 0,
    int64_t v_head_stride = 0,
    int64_t o_batch_stride = 0,
    int64_t o_row_stride = 0,
    int64_t o_head_stride = 0,

    // Scaling parameters
    float softmax_scale = 0.0f,
```

```
267         float p_dropout = 0.0f,
268         float softcap = 0.0f,
269 
270         // Masking parameters
271         bool is_causal = false,
272         int window_size_left = -1,
273         int window_size_right = -1,
274 
275         // Variable length parameters
276         int* cu_seqlens_q = nullptr,
277         int* cu_seqlens_k = nullptr,
278         int* seqused_k = nullptr,
279         int* leftpad_k = nullptr,
280         bool is_seqlens_k_cumulative = true,
281 
282         // KV cache parameters
283         void* knew_ptr = nullptr,
284         void* vnew_ptr = nullptr,
285         int seqlen_knew = 0,
286         int64_t knew_batch_stride = 0,
287         int64_t knew_row_stride = 0,
288         int64_t knew_head_stride = 0,
289         int64_t vnew_batch_stride = 0,
290         int64_t vnew_row_stride = 0,
291         int64_t vnew_head_stride = 0,
292 
293         // Rotary embedding parameters
294         void* rotary_cos_ptr = nullptr,
295         void* rotary_sin_ptr = nullptr,
296         int rotary_dim = 0,
297         bool is_rotary_interleaved = true,
298 
299         // Cache indexing
300         int* cache_batch_idx = nullptr,
301 
302         // Paged KV cache parameters
303         int* block_table = nullptr,
304         int64_t block_table_batch_stride = 0,
305         int page_block_size = 0,
306 
307         // ALiBi parameters
308         void* alibi_slopes_ptr = nullptr,
309         int64_t alibi_slopes_batch_stride = 0,
310 
311         // Split computation parameters
312         int num_splits = 0,
313         void* oaccum_ptr = nullptr,
```

```
    void* softmax_lseaccum_ptr = nullptr,

    // Miscellaneous flags
    bool is_bf16 = false,
    bool unpadded_lse = false,
    bool seqlenq_ngroups_swapped = false,

    // Execution parameters
    cudaStream_t stream = nullptr,
    bool force_split_kernel = false
    // Return options
    bool return_softmax_lse = false
);
```

With kvcache

代码块

```
/**
 * @brief Executes Flash Attention forward pass with KV cache
 *
 * @param dev             Device handle for CUDA/CPU device management
 *                        Used to identify the device where tensors are allocated and computation will occur
 *
 * @param q_ptr           Pointer to query tensor data (batch_size, seqlen_q, nheads, headdim)
 *                        Current query tokens to compute attention for
 *                        For decoder scenarios, typically has seqlen_q=1 for autoregressive generation
 *
 * @param k_cache_ptr     Pointer to key cache tensor data
 *                        For standard KV cache: (batch_size_cache, seqlen_cache, nheads_k, headdim)
 *                        For paged KV cache: (num_blocks, block_size, nheads_k, headdim)
 *                        Contains previously processed keys
 *
 * @param v_cache_ptr     Pointer to value cache tensor data
 *                        For standard KV cache: (batch_size_cache, seqlen_cache, nheads_k, headdim)
 *                        For paged KV cache: (num_blocks, block_size, nheads_k, headdim)
 *                        Contains previously processed values
 *
 * @param o_ptr           Pointer to output tensor data (batch_size, seqlen_q, nheads, headdim)
```

```
 *                          Will store the result of attention computation
 *
 * @param softmax_lse_ptr  Pointer to store log-sum-exp values (batch_size, nheads, seqlen_q)
 *                          Used for numerical stability. Set to nullptr if not needed.
 *
 * @param batch_size       Number of sequences in the batch
 *                          For inference scenarios, often set to 1
 *
 * @param seqlen_q         Length of each query sequence
 *                          For autoregressive generation, typically 1
 *
 * @param seqlen_cache     Maximum sequence length in the KV cache
 *                          Represents the capacity of the cache, not necessarily fully used
 *
 * @param num_heads        Number of attention heads for query tensor
 *                          Each head performs independent attention computation
 *
 * @param num_heads_k      Number of attention heads for key/value tensors
 *                          For standard attention: equals num_heads
 *                          For multi-query attention (MQA): equals 1
 *                          For grouped-query attention (GQA): equals num_heads / group_size
 *
 * @param head_size        Dimension of each attention head
 *                          Typically 64, 80, or 128 in modern architectures
 *                          Must be ≤ 256 for Flash Attention implementation
 *
 * @param q_batch_stride   Stride (in elements) between consecutive batches in query tensor
 *                          For contiguous tensor: seqlen_q * num_heads * head_size
 *
 * @param q_row_stride     Stride (in elements) between consecutive rows in query tensor
 *                          For contiguous tensor: num_heads * head_size
 *
 * @param q_head_stride    Stride (in elements) between consecutive heads in query tensor
 *                          For contiguous tensor: head_size
 *
 * @param k_cache_batch_stride Stride between batches in key cache tensor
 *                              For contiguous tensor: seqlen_cache * num_heads_k * head_size
 *                              Ignored for paged KV cache
```

```
60   *
61   * @param k_cache_row_stride   Stride between rows in key cache tensor
62   *                             For contiguous tensor: num_heads_k * head_size
63   *                             For paged KV cache: block_size * num_heads_k *
     head_size
64   *
65   * @param k_cache_head_stride  Stride between heads in key cache tensor
66   *                             For contiguous tensor: head_size
67   *
68   * @param v_cache_batch_stride Stride between batches in value cache tensor
69   *                             For contiguous tensor: seqlen_cache *
     num_heads_k * head_size
70   *                             Ignored for paged KV cache
71   *
72   * @param v_cache_row_stride   Stride between rows in value cache tensor
73   *                             For contiguous tensor: num_heads_k * head_size
74   *                             For paged KV cache: block_size * num_heads_k *
     head_size
75   *
76   * @param v_cache_head_stride  Stride between heads in value cache tensor
77   *                             For contiguous tensor: head_size
78   *
79   * @param o_batch_stride   Stride between batches in output tensor
80   *                         For contiguous tensor: seqlen_q * num_heads *
     head_size
81   *
82   * @param o_row_stride     Stride between rows in output tensor
83   *                         For contiguous tensor: num_heads * head_size
84   *
85   * @param o_head_stride    Stride between heads in output tensor
86   *                         For contiguous tensor: head_size
87   *
88   * @param softmax_scale    Scale factor applied to attention scores before
     softmax
89   *                         Typically 1/sqrt(head_size) to normalize variance
90   *                         Set to 0.0 to use default scale of 1/sqrt(head_size)
91   *
92   * @param softcap          Softcapping value for attention scores (0.0 to
     disable)
93   *                         If > 0.0, applies a soft cap to attention scores to
     prevent extreme values
94   *
95   * @param is_causal        Whether to apply causal attention mask
96   *                         If true, each position can only attend to previous
     positions (including itself)
97   *                         Used in autoregressive models to prevent looking at
     future tokens
```

```
 98    *
 99    * @param window_size_left Left window size for local attention (-1 for
       unlimited)
100    *                         If >= 0, limits how far back each query can attend
101    *
102    * @param window_size_right Right window size for local attention (-1 for
       unlimited)
103    *                          If >= 0, limits how far forward each query can
       attend
104    *                          Only relevant when is_causal=false
105    *
106    * @param k_new_ptr        New key tensor data pointer (batch_size,
       seqlen_knew, nheads_k, headdim)
107    *                         Contains new keys to be appended to the KV cache
108    *                         Set to nullptr if not appending to cache
109    *
110    * @param v_new_ptr        New value tensor data pointer (batch_size,
       seqlen_knew, nheads_k, headdim)
111    *                         Contains new values to be appended to the KV cache
112    *                         Set to nullptr if not appending to cache
113    *
114    * @param seqlen_knew      Length of new keys/values to append to the cache
115    *                         Typically 1 for autoregressive generation
116    *
117    * @param k_new_batch_stride Stride between batches in new key tensor
118    *                           For contiguous tensor: seqlen_knew * num_heads_k
       * head_size
119    *
120    * @param k_new_row_stride   Stride between rows in new key tensor
121    *                           For contiguous tensor: num_heads_k * head_size
122    *
123    * @param k_new_head_stride  Stride between heads in new key tensor
124    *                           For contiguous tensor: head_size
125    *
126    * @param v_new_batch_stride Stride between batches in new value tensor
127    *                           For contiguous tensor: seqlen_knew * num_heads_k
       * head_size
128    *
129    * @param v_new_row_stride   Stride between rows in new value tensor
130    *                           For contiguous tensor: num_heads_k * head_size
131    *
132    * @param v_new_head_stride  Stride between heads in new value tensor
133    *                           For contiguous tensor: head_size
134    *
135    * @param cache_seqlens    Current sequence lengths in KV cache (batch_size)
136    *                         Specifies the current length of each sequence in
       the cache
```

```
137     *                         Required when appending new keys/values to the cache
138     *
139     * @param rotary_cos_ptr   Cosine table for rotary embeddings (seqlen_ro,
    rotary_dim/2)
140     *                         Contains precomputed cosine values for rotary
    position embeddings
141     *                         seqlen_ro should be >= maximum sequence length. Set
    to nullptr to disable.
142     *
143     * @param rotary_sin_ptr   Sine table for rotary embeddings (seqlen_ro,
    rotary_dim/2)
144     *                         Contains precomputed sine values for rotary
    position embeddings
145     *                         seqlen_ro should be >= maximum sequence length. Set
    to nullptr to disable.
146     *
147     * @param rotary_dim       Dimension for rotary embeddings (must be <=
    head_size and divisible by 16)
148     *                         Only the first rotary_dim dimensions of each head
    will have rotary embeddings
149     *                         Set to 0 if not using rotary embeddings
150     *
151     * @param is_rotary_interleaved Whether rotary embeddings are interleaved
152     *                              If true, dimensions are paired as (0,1),
    (2,3), etc. (GPT-J style)
153     *                              If false, dimensions are paired as
    (0,rotary_dim/2), (1,rotary_dim/2+1), etc. (GPT-NeoX style)
154     *
155     * @param cache_batch_idx  Indices for batch mapping (batch_size)
156     *                         Maps from logical batch indices to physical indices
    in the KV cache
157     *                         Useful for reordering or sharing KV cache across
    requests. Set to nullptr for direct mapping.
158     *
159     * @param cache_leftpad    Left padding for each sequence in cache (batch_size)
160     *                         Specifies number of padding tokens at the beginning
    of each sequence
161     *                         Affects position calculations for rotary
    embeddings. Set to nullptr if no left padding.
162     *
163     * @param block_table      Block table for paged KV cache (batch_size,
    max_blocks_per_seq)
164     *                         Contains block indices for each sequence in paged
    KV cache
165     *                         Each row maps logical sequence positions to
    physical memory blocks. Set to nullptr for contiguous KV cache.
166     *
```

```
167     * @param block_table_batch_stride Stride between batches in block table
168     *                                 For contiguous tensor: max_blocks_per_seq
169     *
170     * @param page_block_size  Size of each block in paged KV cache
171     *                         Typically 16 or 32 tokens per block
172     *                         Only relevant when using paged KV cache
    (block_table != nullptr)
173     *
174     * @param alibi_slopes_ptr ALiBi position bias slopes (nheads) or (batch_size,
    nheads)
175     *                         Contains slope values for ALiBi positional encoding
176     *                         Different slopes per head create different
    attention patterns. Set to nullptr to disable ALiBi.
177     *
178     * @param alibi_slopes_batch_stride Stride between batches in ALiBi slopes
179     *                                  For contiguous tensor: nheads
180     *                                  Set to 0 if alibi_slopes is shared across
    batch (shape is just (nheads))
181     *
182     * @param num_splits       Number of splits for computation (0 for auto)
183     *                         Splits the key/value tensors into chunks along
    sequence dimension
184     *                         Helps with memory efficiency for long sequences. 0
    means auto-select based on GPU.
185     *
186     * @param oaccum_ptr       Accumulated output pointer for split computation
187     *                         Used internally for split-KV optimization
188     *                         Set to nullptr unless manually implementing split
    computation
189     *
190     * @param softmax_lseaccum_ptr Accumulated LSE pointer for split computation
191     *                             Used internally for split-KV optimization
192     *                             Set to nullptr unless manually implementing
    split computation
193     *
194     * @param is_bf16          Whether tensors are in BF16 format (true) or FP16
    format (false)
195     *                         BF16 has less precision but more range than FP16
196     *                         Must accurately reflect the data type of input
    tensors
197     *
198     * @param stream           CUDA stream to execute the kernel on
199     *                         Allows asynchronous execution and integration with
    other CUDA operations
200     *                         Set to nullptr to use the current CUDA stream
201     *
202     * @param return_softmax_lse Whether to return the log-sum-exp values
```

```
 *                          If true, softmax_lse_ptr must be a valid pointer
 *
 * @return Status code (0 for success, non-zero for failure)
 */
int run_mha_fwd_kvcache(
    void* dev,                          // Device handle

    // Core tensor pointers
    void* q_ptr,
    void* k_cache_ptr,
    void* v_cache_ptr,
    void* o_ptr,
    void* softmax_lse_ptr = nullptr,

    // Dimensions
    int batch_size = 1,
    int seqlen_q = 1,
    int seqlen_cache = 0,
    int num_heads = 0,
    int num_heads_k = 0,
    int head_size = 0,

    // Stride information (in elements, not bytes)
    int64_t q_batch_stride = 0,
    int64_t q_row_stride = 0,
    int64_t q_head_stride = 0,
    int64_t k_cache_batch_stride = 0,
    int64_t k_cache_row_stride = 0,
    int64_t k_cache_head_stride = 0,
    int64_t v_cache_batch_stride = 0,
    int64_t v_cache_row_stride = 0,
    int64_t v_cache_head_stride = 0,
    int64_t o_batch_stride = 0,
    int64_t o_row_stride = 0,
    int64_t o_head_stride = 0,

    // Scaling parameters
    float softmax_scale = 0.0f,
    float softcap = 0.0f,

    // Masking parameters
    bool is_causal = true,
    int window_size_left = -1,
    int window_size_right = -1,

    // New KV parameters
    void* k_new_ptr = nullptr,
```

```
250         void* v_new_ptr = nullptr,
251         int seqlen_knew = 0,
252         int64_t k_new_batch_stride = 0,
253         int64_t k_new_row_stride = 0,
254         int64_t k_new_head_stride = 0,
255         int64_t v_new_batch_stride = 0,
256         int64_t v_new_row_stride = 0,
257         int64_t v_new_head_stride = 0,
258
259         // KV cache parameters
260         int* cache_seqlens = nullptr,
261
262         // Rotary embedding parameters
263         void* rotary_cos_ptr = nullptr,
264         void* rotary_sin_ptr = nullptr,
265         int rotary_dim = 0,
266         bool is_rotary_interleaved = true,
267
268         // Cache indexing
269         int* cache_batch_idx = nullptr,
270         int* cache_leftpad = nullptr,
271
272         // Paged KV cache parameters
273         int* block_table = nullptr,
274         int64_t block_table_batch_stride = 0,
275         int page_block_size = 0,
276
277         // ALiBi parameters
278         void* alibi_slopes_ptr = nullptr,
279         int64_t alibi_slopes_batch_stride = 0,
280
281         // Split computation parameters
282         int num_splits = 0,
283         void* oaccum_ptr = nullptr,
284         void* softmax_lseaccum_ptr = nullptr,
285
286         // Data type flags
287         bool is_bf16 = false,
288
289         // Execution parameters
290         cudaStream_t stream = nullptr,
291
292         // Return options
293         bool return_softmax_lse = false
294     );
```

~~FA3~~

~~Without kvcache~~

代码块

```
/**
 * @brief Executes the Flash Attention 3 forward pass with fully explicit parameters
 *
 * @param dev               Device handle for CUDA/CPU device management
 *                          Used to identify the device where tensors are allocated and computation will occur
 *
 * @param q_ptr             Pointer to query tensor data (total_q, nheads, headdim) or (batch_size, seqlen_q, nheads, headdim)
 *                          For variable length sequences, total_q = sum of all sequence lengths in batch
 *
 * @param k_ptr             Pointer to key tensor data (total_k, nheads_k, headdim) or (batch_size, seqlen_k, nheads_k, headdim)
 *                          For KV cache, this points to the cached keys
 *
 * @param v_ptr             Pointer to value tensor data (total_k, nheads_k, headdim_v) or (batch_size, seqlen_k, nheads_k, headdim_v)
 *                          For KV cache, this points to the cached values
 *                          Note: headdim_v can be different from headdim in FA3
 *
 * @param o_ptr             Pointer to output tensor data (total_q, nheads, headdim_v) or (batch_size, seqlen_q, nheads, headdim_v)
 *                          Will store the result of attention computation
 *
 * @param softmax_lse_ptr   Pointer to store log-sum-exp values from softmax computation (nheads, total_q) or (batch_size, nheads, seqlen_q)
 *                          Used for numerical stability and backward pass. Set to nullptr if not needed.
 *
 * @param batch_size        Number of sequences in the batch
 *                          For fixed length: all sequences have the same length
 *                          For variable length: sequences can have different lengths
 *
 * @param seqlen_q          Length of each query sequence (for fixed length) or maximum query sequence length (for variable length)
 *                          In decoder scenarios, typically 1 for autoregressive generation
 *
```

```
30   * @param seqlen_k          Length of each key/value sequence (for fixed
     length) or maximum key/value sequence length (for variable length)
31   *                         In KV cache scenarios, this is the current length
     of the cached context
32   *
33   * @param num_heads         Number of attention heads for query tensor
34   *                         Each head performs independent attention computation
35   *
36   * @param num_heads_k       Number of attention heads for key/value tensors
37   *                         For standard attention: equals num_heads
38   *                         For multi-query attention (MQA): equals 1
39   *                         For grouped-query attention (GQA): equals num_heads
     / group_size
40   *
41   * @param head_size         Dimension of each attention head for query/key
42   *                         Typically 64, 80, or 128 in modern architectures
43   *                         Must be ≤ 256 for Flash Attention implementation
44   *
45   * @param head_size_v       Dimension of each attention head for value/output
46   *                         Can be different from head_size in FA3
47   *                         Set to 0 to use the same value as head_size
48   *
49   * @param q_batch_stride    Stride (in elements) between consecutive batches in
     query tensor
50   *                         For contiguous tensor: seqlen_q * num_heads *
     head_size
51   *
52   * @param q_row_stride      Stride (in elements) between consecutive rows
     (sequences) in query tensor
53   *                         For contiguous tensor: num_heads * head_size
54   *
55   * @param q_head_stride     Stride (in elements) between consecutive heads in
     query tensor
56   *                         For contiguous tensor: head_size
57   *
58   * @param k_batch_stride    Stride (in elements) between consecutive batches in
     key tensor
59   *                         For contiguous tensor: seqlen_k * num_heads_k *
     head_size
60   *
61   * @param k_row_stride      Stride (in elements) between consecutive rows
     (sequences) in key tensor
62   *                         For contiguous tensor: num_heads_k * head_size
63   *
64   * @param k_head_stride     Stride (in elements) between consecutive heads in
     key tensor
65   *                         For contiguous tensor: head_size
```

```
66    *
67    * @param v_batch_stride   Stride (in elements) between consecutive batches in
      value tensor
68    *                         For contiguous tensor: seqlen_k * num_heads_k *
      head_size_v
69    *
70    * @param v_row_stride     Stride (in elements) between consecutive rows
      (sequences) in value tensor
71    *                         For contiguous tensor: num_heads_k * head_size_v
72    *
73    * @param v_head_stride    Stride (in elements) between consecutive heads in
      value tensor
74    *                         For contiguous tensor: head_size_v
75    *
76    * @param v_dim_stride     Stride (in elements) between consecutive dimensions
      in value tensor
77    *                         For row-major: 1, for column-major: v_head_stride
78    *                         FA3 supports column-major layout for value tensor
79    *
80    * @param o_batch_stride   Stride (in elements) between consecutive batches in
      output tensor
81    *                         For contiguous tensor: seqlen_q * num_heads *
      head_size_v
82    *
83    * @param o_row_stride     Stride (in elements) between consecutive rows
      (sequences) in output tensor
84    *                         For contiguous tensor: num_heads * head_size_v
85    *
86    * @param o_head_stride    Stride (in elements) between consecutive heads in
      output tensor
87    *                         For contiguous tensor: head_size_v
88    *
89    * @param softmax_scale    Scale factor applied to attention scores before
      softmax
90    *                         Typically 1/sqrt(head_size) to normalize variance
91    *                         Affects the sharpness of attention distribution
92    *
93    * @param softcap          Softcapping value for attention scores (0.0 to
      disable)
94    *                         If > 0.0, applies a soft cap to attention scores to
      prevent extreme values
95    *                         Helps with training stability for very long
      sequences
96    *
97    * @param is_causal        Whether to apply causal attention mask
98    *                         If true, each position can only attend to previous
      positions (including itself)
```

```
 99   *                      Used in autoregressive models to prevent looking at
      future tokens
100   *
101   * @param window_size_left Left window size for local attention (-1 for
      unlimited)
102   *                        If >= 0, limits how far back each query can attend
103   *                        Useful for long sequence efficiency or implementing
      sliding window attention
104   *
105   * @param window_size_right Right window size for local attention (-1 for
      unlimited)
106   *                         If >= 0, limits how far forward each query can
      attend
107   *                         Only relevant when is_causal=false
108   *
109   * @param cu_seqlens_q     Cumulative sequence lengths for queries
      (batch_size+1)
110   *                        Format: [0, len1, len1+len2, len1+len2+len3, ...]
111   *                        Used for variable length sequences. Set to nullptr
      for fixed length.
112   *
113   * @param cu_seqlens_k     Cumulative sequence lengths for keys/values
      (batch_size+1)
114   *                        Format: [0, len1, len1+len2, len1+len2+len3, ...]
115   *                        Used for variable length sequences. Set to nullptr
      for fixed length.
116   *
117   * @param cu_seqlens_knew  Cumulative sequence lengths for new keys/values
      (batch_size+1)
118   *                        Used when appending to KV cache with variable
      length sequences
119   *
120   * @param seqused_q        Actual sequence usage for queries (batch_size)
121   *                        Specifies how much of each sequence is actually used
122   *                        New in FA3: allows specifying active sequence
      lengths
123   *
124   * @param seqused_k        Actual sequence usage for keys (batch_size)
125   *                        Specifies how much of each sequence in the KV cache
      is actually used
126   *                        Useful when allocated length exceeds used length
127   *
128   * @param leftpad_k        Left padding for each sequence in key (batch_size)
129   *                        Specifies number of padding tokens at the beginning
      of each sequence
130   *                        Affects position calculations for rotary embeddings
131   *
```

```
132  * @param qv_ptr           Pointer to fused QV tensor for QV attention
     (total_q, nheads, headdim_v) or nullptr
133  *                         New in FA3: allows fused QV computation for certain
     architectures
134  *
135  * @param qv_batch_stride  Stride between batches in QV tensor
136  * @param qv_row_stride    Stride between rows in QV tensor
137  * @param qv_head_stride   Stride between heads in QV tensor
138  *
139  * @param q_descale_ptr    Pointer to query descaling factors (batch_size,
     nheads) or nullptr
140  *                         Used for FP8 precision scaling in FA3
141  *
142  * @param k_descale_ptr    Pointer to key descaling factors (batch_size,
     nheads_k) or nullptr
143  *                         Used for FP8 precision scaling in FA3
144  *
145  * @param v_descale_ptr    Pointer to value descaling factors (batch_size,
     nheads_k) or nullptr
146  *                         Used for FP8 precision scaling in FA3
147  *
148  * @param q_descale_batch_stride Stride between batches in query descaling
     factors
149  * @param q_descale_head_stride  Stride between heads in query descaling
     factors
150  * @param k_descale_batch_stride Stride between batches in key descaling
     factors
151  * @param k_descale_head_stride  Stride between heads in key descaling factors
152  * @param v_descale_batch_stride Stride between batches in value descaling
     factors
153  * @param v_descale_head_stride  Stride between heads in value descaling
     factors
154  *
155  * @param knew_ptr         New key tensor data pointer (batch_size,
     seqlen_knew, nheads_k, headdim)
156  *                         Contains new keys to be appended to the KV cache
157  *
158  * @param vnew_ptr         New value tensor data pointer (batch_size,
     seqlen_knew, nheads_k, headdim_v)
159  *                         Contains new values to be appended to the KV cache
160  *
161  * @param seqlen_knew      Length of new keys/values to append to the cache
162  *                         Typically 1 for autoregressive generation, or chunk
     size for batch processing
163  *
164  * @param knew_batch_stride Stride between batches in new key tensor
165  * @param knew_row_stride   Stride between rows in new key tensor
```

```
166   * @param knew_head_stride  Stride between heads in new key tensor
167   * @param vnew_batch_stride Stride between batches in new value tensor
168   * @param vnew_row_stride   Stride between rows in new value tensor
169   * @param vnew_head_stride  Stride between heads in new value tensor
170   *
171   * @param rotary_cos_ptr    Cosine table for rotary embeddings (seqlen_ro,
      rotary_dim/2)
172   *                          Contains precomputed cosine values for rotary
      position embeddings
173   *
174   * @param rotary_sin_ptr    Sine table for rotary embeddings (seqlen_ro,
      rotary_dim/2)
175   *                          Contains precomputed sine values for rotary
      position embeddings
176   *
177   * @param rotary_dim        Dimension for rotary embeddings (must be <=
      head_size and divisible by 16)
178   *                          Only the first rotary_dim dimensions of each head
      will have rotary embeddings
179   *
180   * @param is_rotary_interleaved Whether rotary embeddings are interleaved
181   *                              If true, dimensions are paired as (0,1),
      (2,3), etc. (GPT-J style)
182   *                              If false, dimensions are paired as
      (0,rotary_dim/2), (1,rotary_dim/2+1), etc. (GPT-NeoX style)
183   *
184   * @param kv_batch_idx      Indices for batch mapping (batch_size)
185   *                          Maps from logical batch indices to physical
      indices in the KV cache
186   *
187   * @param page_table        Block table for paged KV cache (batch_size,
      max_blocks_per_seq)
188   *                          Contains block indices for each sequence in paged
      KV cache
189   *
190   * @param page_table_batch_stride Stride between batches in block table
191   * @param page_size         Size of each page in paged KV cache
192   * @param num_pages         Total number of pages in paged KV cache
193   * @param pagedkv_tma       Whether to use TMA (Tensor Memory Access) for
      paged KV cache
194   *                          TMA is a hardware feature in NVIDIA Hopper
      architecture
195   *
196   * @param num_splits        Number of splits for computation (0 for auto)
197   *                          Splits the key/value tensors into chunks along
      sequence dimension
198   *
```

```
199     * @param pack_gqa          Whether to pack grouped-query attention for
    optimization
200     *                         New in FA3: allows specialized GQA implementation
201     *
202     * @param scheduler_metadata Pointer to scheduler metadata for varlen
    optimization
203     *                          New in FA3: allows precomputed scheduling
    information
204     *
205     * @param sm_margin         Number of SMs to reserve for communication (0 for
    none)
206     *                         New in FA3: allows reserving SMs for communication
    in multi-GPU scenarios
207     *
208     * @param is_bf16           Whether tensors are in BF16 format (true) or FP16
    format (false)
209     * @param is_fp32           Whether tensors are in FP32 format
210     * @param is_e4m3           Whether tensors are in E4M3 (FP8) format
211     *
212     * @param stream            CUDA stream to execute the kernel on
213     *
214     * @return Status code (0 for success, non-zero for failure)
215     */
216    int run_mha_fwd(
217        void* dev,                      // Device handle
218
219        // Core tensor pointers
220        void* q_ptr,
221        void* k_ptr,
222        void* v_ptr,
223        void* o_ptr,
224        void* softmax_lse_ptr = nullptr,
225
226        // Dimensions
227        int batch_size = 1,
228        int seqlen_q = 0,
229        int seqlen_k = 0,
230        int num_heads = 0,
231        int num_heads_k = 0,
232        int head_size = 0,
233        int head_size_v = 0,             // New in FA3: can be different from
    head_size
234
235        // Stride information (in elements, not bytes)
236        int64_t q_batch_stride = 0,
237        int64_t q_row_stride = 0,
238        int64_t q_head_stride = 0,
```

```
239       int64_t k_batch_stride = 0,
240       int64_t k_row_stride = 0,
241       int64_t k_head_stride = 0,
242       int64_t v_batch_stride = 0,
243       int64_t v_row_stride = 0,
244       int64_t v_head_stride = 0,
245       int64_t v_dim_stride = 1,         // New in FA3: supports column-major
    layout
246       int64_t o_batch_stride = 0,
247       int64_t o_row_stride = 0,
248       int64_t o_head_stride = 0,
249
250       // Scaling parameters
251       float softmax_scale = 0.0f,
252       float softcap = 0.0f,
253
254       // Masking parameters
255       bool is_causal = false,
256       int window_size_left = -1,
257       int window_size_right = -1,
258
259       // Variable length parameters
260       int* cu_seqlens_q = nullptr,
261       int* cu_seqlens_k = nullptr,
262       int* cu_seqlens_knew = nullptr,  // New in FA3
263       int* seqused_q = nullptr,        // New in FA3
264       int* seqused_k = nullptr,
265       int* leftpad_k = nullptr,
266
267       // QV fusion parameters (new in FA3)
268       void* qv_ptr = nullptr,
269       int64_t qv_batch_stride = 0,
270       int64_t qv_row_stride = 0,
271       int64_t qv_head_stride = 0,
272
273       // Descaling parameters for FP8 (new in FA3)
274       float* q_descale_ptr = nullptr,
275       float* k_descale_ptr = nullptr,
276       float* v_descale_ptr = nullptr,
277       int64_t q_descale_batch_stride = 0,
278       int64_t q_descale_head_stride = 0,
279       int64_t k_descale_batch_stride = 0,
280       int64_t k_descale_head_stride = 0,
281       int64_t v_descale_batch_stride = 0,
282       int64_t v_descale_head_stride = 0,
283
284       // KV cache parameters
```

```
    void* knew_ptr = nullptr,
    void* vnew_ptr = nullptr,
    int seqlen_knew = 0,
    int64_t knew_batch_stride = 0,
    int64_t knew_row_stride = 0,
    int64_t knew_head_stride = 0,
    int64_t vnew_batch_stride = 0,
    int64_t vnew_row_stride = 0,
    int64_t vnew_head_stride = 0,

    // Rotary embedding parameters
    void* rotary_cos_ptr = nullptr,
    void* rotary_sin_ptr = nullptr,
    int rotary_dim = 0,
    bool is_rotary_interleaved = true,

    // Cache indexing
    int* kv_batch_idx = nullptr,

    // Paged KV cache parameters
    int* page_table = nullptr,
    int64_t page_table_batch_stride = 0,
    int page_size = 0,
    int num_pages = 0,
    bool pagedkv_tma = false,        // New in FA3: TMA support

    // Split computation parameters
    int num_splits = 1,
    bool pack_gqa = false,           // New in FA3
    void* scheduler_metadata = nullptr, // New in FA3
    int sm_margin = 0,               // New in FA3

    // Data type flags
    bool is_bf16 = false,
    bool is_fp32 = false,
    bool is_e4m3 = false,            // New in FA3: FP8 support

    // Execution parameters
    cudaStream_t stream = nullptr
);
```

~~With kvcache~~

代码块

```
/**
 * @brief Executes the Flash Attention 3 forward pass with KV cache
```

```
 3    *
 4    * @param dev              Device handle for CUDA/CPU device management
 5    *
 6    * @param q_ptr            Pointer to query tensor data (batch_size, seqlen_q,
      nheads, headdim)
 7    *                         Current query tokens to compute attention for
 8    *
 9    * @param k_cache_ptr      Pointer to key cache tensor data
10    *                         For standard KV cache: (batch_size_cache,
      seqlen_cache, nheads_k, headdim)
11    *                         For paged KV cache: (num_blocks, page_size,
      nheads_k, headdim)
12    *
13    * @param v_cache_ptr      Pointer to value cache tensor data
14    *                         For standard KV cache: (batch_size_cache,
      seqlen_cache, nheads_k, headdim_v)
15    *                         For paged KV cache: (num_blocks, page_size,
      nheads_k, headdim_v)
16    *
17    * @param o_ptr            Pointer to output tensor data (batch_size,
      seqlen_q, nheads, headdim_v)
18    *                         Will store the result of attention computation
19    *
20    * @param softmax_lse_ptr  Pointer to store log-sum-exp values (batch_size,
      nheads, seqlen_q)
21    *                         Used for numerical stability. Set to nullptr if not
      needed.
22    *
23    * @param batch_size       Number of sequences in the batch
24    * @param seqlen_q         Length of each query sequence (typically 1 for
      autoregressive generation)
25    * @param seqlen_cache     Maximum sequence length in the KV cache
26    * @param num_heads        Number of attention heads for query tensor
27    * @param num_heads_k      Number of attention heads for key/value tensors
      (for MQA/GQA)
28    * @param head_size        Dimension of each attention head for query/key
29    * @param head_size_v      Dimension of each attention head for value/output
      (can be different from head_size)
30    *
31    * @param q_batch_stride   Stride between batches in query tensor
32    * @param q_row_stride     Stride between rows in query tensor
33    * @param q_head_stride    Stride between heads in query tensor
34    * @param k_cache_batch_stride Stride between batches in key cache tensor
      (ignored for paged KV cache)
35    * @param k_cache_row_stride   Stride between rows in key cache tensor
36    * @param k_cache_head_stride  Stride between heads in key cache tensor
```

```
37   * @param v_cache_batch_stride Stride between batches in value cache tensor
     (ignored for paged KV cache)
38   * @param v_cache_row_stride   Stride between rows in value cache tensor
39   * @param v_cache_head_stride  Stride between heads in value cache tensor
40   * @param v_cache_dim_stride   Stride between dimensions in value cache tensor
     (for column-major layout)
41   * @param o_batch_stride   Stride between batches in output tensor
42   * @param o_row_stride     Stride between rows in output tensor
43   * @param o_head_stride    Stride between heads in output tensor
44   *
45   * @param softmax_scale    Scale factor for attention scores
46   * @param softcap          Softcapping value (0.0 to disable)
47   *
48   * @param is_causal        Whether to apply causal attention mask
49   * @param window_size_left Left window size for local attention (-1 for
     unlimited)
50   * @param window_size_right Right window size for local attention (-1 for
     unlimited)
51   *
52   * @param k_new_ptr        New key tensor data pointer (batch_size,
     seqlen_k_new, nheads_k, headdim)
53   *                         Contains new keys to be appended to the KV cache
54   *
55   * @param v_new_ptr        New value tensor data pointer (batch_size,
     seqlen_k_new, nheads_k, headdim_v)
56   *                         Contains new values to be appended to the KV cache
57   *
58   * @param seqlen_k_new     Length of new keys/values to append to the cache
59   *                         Typically 1 for autoregressive generation
60   *
61   * @param k_new_batch_stride Stride between batches in new key tensor
62   * @param k_new_row_stride   Stride between rows in new key tensor
63   * @param k_new_head_stride  Stride between heads in new key tensor
64   * @param v_new_batch_stride Stride between batches in new value tensor
65   * @param v_new_row_stride   Stride between rows in new value tensor
66   * @param v_new_head_stride  Stride between heads in new value tensor
67   *
68   * @param qv_ptr           Pointer to fused QV tensor for QV attention
     (batch_size, seqlen_q, nheads, headdim_v)
69   *                         New in FA3: allows fused QV computation for certain
     architectures
70   *
71   * @param qv_batch_stride  Stride between batches in QV tensor
72   * @param qv_row_stride    Stride between rows in QV tensor
73   * @param qv_head_stride   Stride between heads in QV tensor
74   *
```

```
 75    * @param q_descale_ptr    Pointer to query descaling factors (batch_size,
       nheads) or nullptr
 76    * @param k_descale_ptr    Pointer to key descaling factors (batch_size,
       nheads_k) or nullptr
 77    * @param v_descale_ptr    Pointer to value descaling factors (batch_size,
       nheads_k) or nullptr
 78    *
 79    * @param cache_seqlens    Current sequence lengths in KV cache (batch_size)
 80    *                         Specifies the current length of each sequence in
       the cache
 81    *
 82    * @param rotary_cos_ptr   Cosine table for rotary embeddings (seqlen_ro,
       rotary_dim/2)
 83    * @param rotary_sin_ptr   Sine table for rotary embeddings (seqlen_ro,
       rotary_dim/2)
 84    * @param rotary_dim       Dimension for rotary embeddings (must be <=
       head_size and divisible by 16)
 85    * @param is_rotary_interleaved Whether rotary embeddings are interleaved
 86    *
 87    * @param cache_batch_idx  Indices for batch mapping (batch_size)
 88    *                         Maps from logical batch indices to physical indices
       in the KV cache
 89    *
 90    * @param cache_leftpad    Left padding for each sequence in cache (batch_size)
 91    *                         Specifies number of padding tokens at the beginning
       of each sequence
 92    *
 93    * @param page_table       Block table for paged KV cache (batch_size,
       max_blocks_per_seq)
 94    *                         Contains block indices for each sequence in paged
       KV cache
 95    *
 96    * @param page_table_batch_stride Stride between batches in block table
 97    * @param page_size        Size of each page in paged KV cache
 98    * @param pagedkv_tma      Whether to use TMA (Tensor Memory Access) for paged
       KV cache
 99    *
100    * @param scheduler_metadata Pointer to scheduler metadata for varlen
       optimization
101    *
102    * @param num_splits       Number of splits for computation (0 for auto)
103    * @param pack_gqa         Whether to pack grouped-query attention for
       optimization
104    * @param sm_margin        Number of SMs to reserve for communication (0 for
       none)
105    *
```

```
106     * @param is_bf16          Whether tensors are in BF16 format (true) or FP16
    format (false)
107     * @param is_fp32          Whether tensors are in FP32 format
108     * @param is_e4m3          Whether tensors are in E4M3 (FP8) format
109     *
110     * @param stream           CUDA stream to execute the kernel on
111     *
112     * @param return_softmax_lse Whether to return the log-sum-exp values
113     *
114     * @return Status code (0 for success, non-zero for failure)
115     */
116    int run_mha_fwd_kvcache(
117        void* dev,                        // Device handle
118
119        // Core tensor pointers
120        void* q_ptr,
121        void* k_cache_ptr,
122        void* v_cache_ptr,
123        void* o_ptr,
124        void* softmax_lse_ptr = nullptr,
125
126        // Dimensions
127        int batch_size = 1,
128        int seqlen_q = 1,
129        int seqlen_cache = 0,
130        int num_heads = 0,
131        int num_heads_k = 0,
132        int head_size = 0,
133        int head_size_v = 0,
134
135        // Stride information (in elements, not bytes)
136        int64_t q_batch_stride = 0,
137        int64_t q_row_stride = 0,
138        int64_t q_head_stride = 0,
139        int64_t k_cache_batch_stride = 0,
140        int64_t k_cache_row_stride = 0,
141        int64_t k_cache_head_stride = 0,
142        int64_t v_cache_batch_stride = 0,
143        int64_t v_cache_row_stride = 0,
144        int64_t v_cache_head_stride = 0,
145        int64_t v_cache_dim_stride = 1,
146        int64_t o_batch_stride = 0,
147        int64_t o_row_stride = 0,
148        int64_t o_head_stride = 0,
149
150        // Scaling parameters
151        float softmax_scale = 0.0f,
```

```
152       float softcap = 0.0f,
153
154       // Masking parameters
155       bool is_causal = false,
156       int window_size_left = -1,
157       int window_size_right = -1,
158
159       // New KV parameters
160       void* k_new_ptr = nullptr,
161       void* v_new_ptr = nullptr,
162       int seqlen_k_new = 0,
163       int64_t k_new_batch_stride = 0,
164       int64_t k_new_row_stride = 0,
165       int64_t k_new_head_stride = 0,
166       int64_t v_new_batch_stride = 0,
167       int64_t v_new_row_stride = 0,
168       int64_t v_new_head_stride = 0,
169
170       // QV fusion parameters
171       void* qv_ptr = nullptr,
172       int64_t qv_batch_stride = 0,
173       int64_t qv_row_stride = 0,
174       int64_t qv_head_stride = 0,
175
176       // Descaling parameters for FP8
177       float* q_descale_ptr = nullptr,
178       float* k_descale_ptr = nullptr,
179       float* v_descale_ptr = nullptr,
180
181       // KV cache parameters
182       int* cache_seqlens = nullptr,
183
184       // Rotary embedding parameters
185       void* rotary_cos_ptr = nullptr,
186       void* rotary_sin_ptr = nullptr,
187       int rotary_dim = 0,
188       bool is_rotary_interleaved = true,
189
190       // Cache indexing
191       int* cache_batch_idx = nullptr,
192       int* cache_leftpad = nullptr,
193
194       // Paged KV cache parameters
195       int* page_table = nullptr,
196       int64_t page_table_batch_stride = 0,
197       int page_size = 0,
198       bool pagedkv_tma = false,
```

```
199
200         // Scheduler metadata
201         void* scheduler_metadata = nullptr,
202
203         // Split computation parameters
204         int num_splits = 0,
205         bool pack_gqa = false,
206         int sm_margin = 0,
207
208         // Data type flags
209         bool is_bf16 = false,
210         bool is_fp32 = false,
211         bool is_e4m3 = false,
212
213         // Execution parameters
214         cudaStream_t stream = nullptr,
215
216         // Return options
217         bool return_softmax_lse = false
218     );
```

ISSUE

- [ ] 传入 cuda kernel 之前会将 tensor 处理为 contiguous

代码块

```
1  @staticmethod
2  def forward(ctx, q, k, v, cu_seqlens_q, cu_seqlens_k, max_seqlen_q,
   max_seqlen_k, ...):
3      # ...
4      q, k, v = [maybe_contiguous(x) for x in (q, k, v)]
5      # ...
```

- [ ] cuda FA2 支持 FP16、BF16 和 FP32 浮点数据类型，
- [ ] cuda FA3（SM90+架构），额外支持 E4M3 （FP8 格式，4 位指数，3 位尾数）

_C::fused_add_rms_norm[vLLM]

参考信息

vLLM 对应的接口实现：https://github.com/vllm-project/vllm/blob/b0ccfc565ac263b4097175b3ba33ad1d5dc5df8a/csrc/layernorm_kernels.cu#L52

Function 接口

代码块

```
/**
 * @brief Performs fused add and RMS normalization
 *
 * @param dev              Device handle for CUDA/CPU device management
 * @param input            First input tensor
 * @param residual         Second input tensor to be added to the first
 * @param weight           Scale factors applied after normalization
 * @param out_sizes        Output tensor sizes
 * @param out_strides      Output tensor strides
 * @param input_sizes      First input tensor sizes
 * @param input_strides    First input tensor strides
 * @param residual_sizes   Second input tensor sizes
 * @param residual_strides Second input tensor strides
 * @param weight_sizes     Weight tensor sizes
 * @param weight_strides   Weight tensor strides
 * @param ndims            Number of dimensions in the tensors
 * @param epsilon          Small constant added for numerical stability
 * @param num_tokens       Number of tokens in the input tensor
 * @param hidden_size      Size of the last dimension in the input tensor
 * @param out_dtype        Data type of output tensor
 * @param input_dtype      Data type of input tensors
 * @param residual_dtype   Data type of residual tensor
 * @param weight_dtype     Data type of weight tensor
 *
 * @note This function:
 *       1. Adds input and residual tensors
 *       2. Computes RMS normalization on the sum
 *       3. Applies weight scaling
 *       4. Stores the result in the input tensor
 *       5. Optionally updates the residual tensor with the sum
 */
void fused_add_rms_norm_kernel(
    void* dev,                      // Device handle
    void* input,                    // First input tensor data pointer
    void* residual,                 // Second input tensor data pointer
    const void* weight,             // Weight tensor data pointer
    const int64_t* input_sizes,     // First input tensor sizes
    const int64_t* input_strides,   // First input tensor strides
    const int64_t* residual_sizes,  // Second input tensor sizes
    const int64_t* residual_strides,// Second input tensor strides
    const int64_t* weight_sizes,    // Weight tensor sizes
    const int64_t* weight_strides,  // Weight tensor strides
    int ndims,                      // Number of dimensions in tensors
    float epsilon,                  // Small constant for numerical stability
```

```
    int num_tokens,                // Number of tokens in the input tensor
    int hidden_size,               // Size of the last dimension in the input
tensor
    int input_dtype,               // Data type of input tensors
    int residual_dtype,            // Data type of residual tensor
    int weight_dtype,              // Data type of weight tensor
    bool inplace_residual          // Whether to update residual in-place
);
```

ISSUE

- [ ] 支持的数据类型 float32 float16 bfloat16
- [ ] 从 vllm/model_executor/layers/layernorm.py 调用看，input，residual， weight 都为 contiguous 布局，但是具体实现也支持非 contiguous

_C::rms_norm[vLLM]

参考信息

实现参考：https://github.com/vllm-project/vllm/blob/cec8c7d7f8753d13737427ceb5cebe987f5f0549/csrc/layernorm_kernels.cu#L17

Function 接口

代码块

```
/**
 * @brief Performs RMS (Root Mean Square) normalization on input tensor
 *
 * @param dev             Device handle for CUDA/CPU device management
 * @param out             Output tensor to store the normalized result
 * @param input           Input tensor to be normalized
 * @param weight          Scale factors applied after normalization
 * @param out_sizes       Output tensor sizes
 * @param out_strides     Output tensor strides (can be non-contiguous)
 * @param input_sizes     Input tensor sizes
 * @param input_strides   Input tensor strides (can be non-contiguous)
 * @param weight_sizes    Weight tensor sizes
 * @param weight_strides  Weight tensor strides
 * @param ndims           Number of dimensions in the input/output tensors
 * @param epsilon         Small constant added for numerical stability
 * @param num_tokens      Number of tokens in the input tensor
 * @param hidden_size     Size of the last dimension in the input tensor
 * @param out_dtype       Data type of output tensor
 * @param input_dtype     Data type of input tensor
```

```
 * @param weight_dtype     Data type of weight tensor
 *
 * @note Memory Layout:
 *       - Fully supports non-contiguous tensors
 *       - Handles arbitrary strides efficiently
 *       - Optimized for both contiguous and non-contiguous cases
 *
 * @note Behavior:
 *       - Normalizes using RMS (sqrt of mean of squares)
 *       - Applies weight scaling after normalization
 *       - Supports different data types for input, output, and weight
 *       - Optimized implementations for different hardware and data types
 */
void rms_norm_kernel(
    void* dev,                    // Device handle
    void* out,                    // Output tensor data pointer
    const void* input,            // Input tensor data pointer
    const void* weight,           // Weight tensor data pointer
    const int64_t* out_sizes,     // Output sizes
    const int64_t* out_strides,   // Output strides
    const int64_t* input_sizes,   // Input tensor sizes
    const int64_t* input_strides, // Input tensor strides
    const int64_t* weight_sizes,  // Weight tensor sizes
    const int64_t* weight_strides,// Weight tensor strides
    int ndims,                    // Number of dimensions in tensors
    const float epsilon,          // Small constant for numerical stability
    int num_tokens,               // Number of tokens in the input tensor
    int hidden_size,              // Size of the last dimension in the input
tensor
    int out_dtype,                // Data type of output tensor
    int input_dtype,              // Data type of input tensor
    int weight_dtype              // Data type of weight tensor
);
```

ISSUE

- [ ] 支持的数据类型 float32 float16 bfloat16
- [ ] 从 vllm/model_executor/layers/layernorm.py 调用看，input，residual， weight 都为 contiguous 布局，但是具体实现也支持非 contiguous

_C::rotary_embedding[vLLM]

参考信息

参考 cuda 实现：https://github.com/vllm-project/vllm/blob/18ed3132d2bfe1df9a74729457b69243955221e8/csrc/pos_encoding_kernels.cu#L124

Function 接口

代码块

```
/**
 * @brief Applies rotary position embeddings to query and key tensors
 *
 * @param dev                 Device handle for CUDA/CPU device management
 * @param positions           Position indices tensor [batch_size, seq_len] or [num_tokens]
 * @param query               Query tensor to apply rotary embeddings to
 * @param key                 Key tensor to apply rotary embeddings to
 * @param cos_sin_cache       Precomputed cosine and sine values [max_position, rot_dim]
 * @param positions_sizes     Sizes of positions tensor
 * @param positions_strides   Strides of positions tensor
 * @param query_sizes         Sizes of query tensor
 * @param query_strides       Strides of query tensor
 * @param key_sizes           Sizes of key tensor
 * @param key_strides         Strides of key tensor
 * @param cache_sizes         Sizes of cos_sin_cache tensor
 * @param cache_strides       Strides of cos_sin_cache tensor
 * @param positions_ndims     Number of dimensions in positions tensor
 * @param query_ndims         Number of dimensions in query tensor
 * @param key_ndims           Number of dimensions in key tensor
 * @param cache_ndims         Number of dimensions in cos_sin_cache tensor
 * @param head_size           Size of each attention head
 * @param is_neox             Whether to use GPT-NeoX style rotary embeddings
 * @param query_dtype         Data type of query tensor
 * @param key_dtype           Data type of key tensor
 * @param cache_dtype         Data type of cos_sin_cache tensor
 *
 * @note Memory Layout:
 *       - Supports both [batch_size, seq_len, num_heads, head_size] and [num_tokens, num_heads, head_size] layouts
 *       - Handles non-contiguous tensors through explicit stride parameters
 *       - Optimized for different data types and hardware
 *
 * @note Behavior:
 *       - Applies rotary position embeddings in-place to query and key tensors
 *       - Supports both GPT-J style (default) and GPT-NeoX style rotary embeddings
```

```
 *       - Efficiently handles grouped-query attention (GQA) where num_heads > num_kv_heads
 */
void rotary_embedding_kernel(
    void* dev,                          // Device handle
    const int64_t* positions,           // Position indices tensor
    void* query,                        // Query tensor
    void* key,                          // Key tensor
    const void* cos_sin_cache,          // Precomputed cosine and sine values
    const int64_t* positions_sizes,     // Sizes of positions tensor
    const int64_t* positions_strides,   // Strides of positions tensor
    const int64_t* query_sizes,         // Sizes of query tensor
    const int64_t* query_strides,       // Strides of query tensor
    const int64_t* key_sizes,           // Sizes of key tensor
    const int64_t* key_strides,         // Strides of key tensor
    const int64_t* cache_sizes,         // Sizes of cos_sin_cache tensor
    const int64_t* cache_strides,       // Strides of cos_sin_cache tensor
    int positions_ndims,                // Number of dimensions in positions tensor
    int query_ndims,                    // Number of dimensions in query tensor
    int key_ndims,                      // Number of dimensions in key tensor
    int cache_ndims,                    // Number of dimensions in cos_sin_cache tensor
    int head_size,                      // Size of each attention head
    bool is_neox,                       // Whether to use GPT-NeoX style rotary embeddings
    int query_dtype,                    // Data type of query tensor
    int key_dtype,                      // Data type of key tensor
    int cache_dtype                     // Data type of cos_sin_cache tensor
);
```

ISSUE

- [ ] 支持的数据类型：float32， float16， bfloat16
- [ ] cuda 的实现有对非 contiguous 的支持

_C::silu_and_mul[vLLM]

参考信息

参考 cuda 实现：https://github.com/vllm-project/vllm/blob/18ed3132d2bfe1df9a74729457b69243955221e8/csrc/activation_kernels.cu#L82

Function 接口

```
/**
 * @brief Applies SiLU activation to first half of input and multiplies with second half
 *
 * @param dev             Device handle for CUDA/CPU device management
 * @param out             Output tensor to store the result
 * @param input           Input tensor with twice the size of output in last dimension
 * @param out_sizes       Output tensor sizes
 * @param out_strides     Output tensor strides (can be non-contiguous)
 * @param input_sizes     Input tensor sizes
 * @param input_strides   Input tensor strides (can be non-contiguous)
 * @param ndims           Number of dimensions in the tensors
 * @param out_dtype       Data type of output tensor
 * @param input_dtype     Data type of input tensor
 *
 * @note Memory Layout:
 *       - Input tensor has shape [..., 2*d] where last dimension contains pairs of values
 *       - Output tensor has shape [..., d]
 *       - Supports non-contiguous tensors through explicit stride parameters
 *
 * @note Behavior:
 *       - Computes out[i] = SiLU(input[i]) * input[i+d] for i in [0,d)
 *       - SiLU(x) = x * sigmoid(x)
 *       - Optimized implementation for different hardware and data types
 */
void silu_and_mul_kernel(
    void* dev,                    // Device handle
    void* out,                    // Output tensor data pointer
    const void* input,            // Input tensor data pointer
    const int64_t* out_sizes,     // Output sizes
    const int64_t* out_strides,   // Output strides
    const int64_t* input_sizes,   // Input tensor sizes
    const int64_t* input_strides, // Input tensor strides
    int ndims,                    // Number of dimensions in tensors
    int out_dtype,                // Data type of output tensor
    int input_dtype               // Data type of input tensor
);
```

ISSUE

- [ ] 支持非 contiguous 输入
- [ ] Cuda kernel 支持的数据类型：float32， float16， bfloat16

首版开发说明

1. 数据类型支持bfloat16，float32和float16，通过template设置
2. 目前llama和deepseek，在dim1_size上已经满足1024B对齐的要求，暂时不考虑不对齐pad的情况
3. dim1_size是output的dimension 1 size。input的dimension 1 size是2*dim1_size

代码块

```
/// @brief Applies SiLU activation to first half of input and multiplies with second half
/// @tparam T Datatype support [sifmt::bfloat16, sifmt::float16, sifmt::float32]
/// @param d_in Input tensor [dim0_size, 2*dim1_size]
/// @param d_out Output tensor [dim0_size, dim1_size]
/// @param dim0_size Size of dimension 0 for the input/output tensor
/// @param dim1_size Size of dimension 1 for the output tensor (total size must be 1024B aligned)
template <typename T>
void silu_and_mul(void *d_in, void *d_out, int dim0_size, int dim1_size);
```

_C_cache_ops::reshape_and_cache_flash[vLLM]

参考信息

reshape_and_cache_flash 是一个用于重塑和缓存注意力机制中的键值对的函数，专为 Flash Attention 优化。通过分析 cache_kernels.cu 中的实现，我可以确定其主要功能：

功能：将模型生成的键（key）和值（value）张量重新排列并存储到 KV 缓存中

数据流：从 [num_tokens， num_heads， head_size] 格式转换为 [num_blocks， block_size， num_heads， head_size] 格式

特殊处理：

支持数据类型转换（如 FP16/BF16 到 FP8）

处理槽位映射（slot mapping）

支持缩放因子（用于量化）

参考实现：https://github.com/vllm-project/vllm/blob/18ed3132d2bfe1df9a74729457b69243955221e8/csrc/cache_kernels.cu#L411

Function 接口

代码块

```

/**
 * @brief Reshapes and caches key-value tensors for Flash Attention
```

```
 4    *
 5    * @param dev              Device handle for CUDA/CPU device management
 6    * @param key              Key tensor [num_tokens, num_heads, head_size]
 7    * @param value            Value tensor [num_tokens, num_heads, head_size]
 8    * @param key_cache        Key cache tensor [num_blocks, block_size,
    num_heads, head_size]
 9    * @param value_cache      Value cache tensor [num_blocks, block_size,
    num_heads, head_size]
10    * @param slot_mapping     Mapping from token indices to cache slots
    [num_tokens]
11    * @param key_sizes        Sizes of key tensor
12    * @param key_strides      Strides of key tensor
13    * @param value_sizes      Sizes of value tensor
14    * @param value_strides    Strides of value tensor
15    * @param key_cache_sizes  Sizes of key cache tensor
16    * @param key_cache_strides Strides of key cache tensor
17    * @param value_cache_sizes Sizes of value cache tensor
18    * @param value_cache_strides Strides of value cache tensor
19    * @param slot_mapping_sizes Sizes of slot mapping tensor
20    * @param slot_mapping_strides Strides of slot mapping tensor
21    * @param key_ndims         Number of dimensions in key tensor
22    * @param value_ndims       Number of dimensions in value tensor
23    * @param key_cache_ndims   Number of dimensions in key cache tensor
24    * @param value_cache_ndims Number of dimensions in value cache tensor
25    * @param slot_mapping_ndims Number of dimensions in slot mapping tensor
26    * @param num_tokens        Number of tokens to process
27    * @param num_heads         Number of attention heads
28    * @param head_size         Size of each attention head
29    * @param block_size        Size of each cache block
30    * @param key_dtype         Data type of key tensor
31    * @param value_dtype       Data type of value tensor
32    * @param key_cache_dtype   Data type of key cache tensor
33    * @param value_cache_dtype Data type of value cache tensor
34    * @param kv_cache_type     Type of KV cache quantization ("fp8", "fp8_e4m3",
    "auto", etc.)
35    * @param k_scale           Scale factor for key quantization [1] or [num_heads]
36    * @param v_scale           Scale factor for value quantization [1] or
    [num_heads]
37    * @param k_scale_size      Size of k_scale tensor (must be 1 or num_heads)
38    * @param v_scale_size      Size of v_scale tensor (must be 1 or num_heads)
39    *
40    * @note Memory Layout:
41    *       - Input tensors (key, value) have shape [num_tokens, num_heads,
    head_size]
42    *       - Cache tensors have shape [num_blocks, block_size, num_heads,
    head_size]
43    *       - Supports non-contiguous tensors through explicit stride parameters
```

```
 *
 * @note Scale Factors:
 *       - k_scale can be either a single value [1] or per-head values [num_heads]
 *       - v_scale can be either a single value [1] or per-head values [num_heads]
 *       - When size is 1, the same scale is applied to all heads
 *       - When size is num_heads, each head uses its own scale factor
 *
 * @note Behavior:
 *       - Reshapes key and value tensors to match the cache format
 *       - Maps tokens to their corresponding cache slots using slot_mapping
 *       - Supports optional quantization to FP8 format with scaling
 *       - Ignores tokens with negative slot indices (padding tokens)
 */
void reshape_and_cache_flash_kernel(
    void* dev,                          // Device handle
    const void* key,                    // Key tensor data pointer
    const void* value,                  // Value tensor data pointer
    void* key_cache,                    // Key cache tensor data pointer
    void* value_cache,                  // Value cache tensor data pointer
    const int64_t* slot_mapping,        // Slot mapping data pointer
    const int64_t* key_sizes,           // Key tensor sizes
    const int64_t* key_strides,         // Key tensor strides
    const int64_t* value_sizes,         // Value tensor sizes
    const int64_t* value_strides,       // Value tensor strides
    const int64_t* key_cache_sizes,     // Key cache tensor sizes
    const int64_t* key_cache_strides,   // Key cache tensor strides
    const int64_t* value_cache_sizes,   // Value cache tensor sizes
    const int64_t* value_cache_strides, // Value cache tensor strides
    const int64_t* slot_mapping_sizes,  // Slot mapping tensor sizes
    const int64_t* slot_mapping_strides,// Slot mapping tensor strides
    int key_ndims,                       // Number of dimensions in key tensor
    int value_ndims,                     // Number of dimensions in value tensor
    int key_cache_ndims,                 // Number of dimensions in key cache tensor
    int value_cache_ndims,               // Number of dimensions in value cache tensor
    int slot_mapping_ndims,              // Number of dimensions in slot mapping tensor
    int num_tokens,                     // Number of tokens to process
    int num_heads,                      // Number of attention heads
    int head_size,                      // Size of each attention head
    int block_size,                     // Size of each cache block
    int key_dtype,                      // Data type of key tensor
    int value_dtype,                    // Data type of value tensor
```

```
    int key_cache_dtype,              // Data type of key cache tensor
    int value_cache_dtype,            // Data type of value cache tensor
    const char* kv_cache_type,        // Type of KV cache quantization
    const float* k_scale,             // Scale factor for key quantization
    const float* v_scale,             // Scale factor for value quantization
    int k_scale_size,                 // Size of k_scale tensor (1 or
num_heads)
    int v_scale_size                  // Size of v_scale tensor (1 or
num_heads)
);
```

ISSUE

- [ ] 支持非 contiguous 输入
- [ ] Cuda kernel 支持的数据类型：float32， float16， bfloat16，量化类型 __NV_E4M3/__NV_E5M2

_C_cache_ops::concat_and_cache_mla

参考信息

concat_and_cache_mla 的核心功能是：

1. 连接（Concatenate）两个张量：

   kv_c：KV 内容张量 [num_tokens， kv_lora_rank]

   k_pe：键位置编码张量 [num_tokens， pe_dim]

2. 将连接后的数据缓存：

   将连接后的数据存储到 KV 缓存中 [num_blocks， block_size， （kv_lora_rank + pe_dim）]

   使用 slot_mapping 将每个 token 映射到缓存中的正确位置

3. 可选的量化处理：

   支持将数据量化为 FP8 格式以节省内存

   使用 scale 参数作为量化的缩放因子

参考实现：https://github.com/vllm-project/vllm/blob/18ed3132d2bfe1df9a74729457b69243955221e8/csrc/cache_kernels.cu#L462

Function 接口

代码块

```
/**
 * @brief Concatenates and caches tensors for Multi-head Latent Attention (MLA)
```

```
 *
 * @param dev                  Device handle for CUDA/CPU device management
 * @param kv_c                 KV content tensor [num_tokens, kv_lora_rank]
 * @param k_pe                 Key position encoding tensor [num_tokens,
pe_dim]
 * @param kv_cache             KV cache tensor [num_blocks, block_size,
(kv_lora_rank + pe_dim)]
 * @param slot_mapping         Mapping from token indices to cache slots
[num_tokens]
 * @param kv_c_sizes           Sizes of KV content tensor
 * @param kv_c_strides         Strides of KV content tensor
 * @param k_pe_sizes           Sizes of key position encoding tensor
 * @param k_pe_strides         Strides of key position encoding tensor
 * @param kv_cache_sizes       Sizes of KV cache tensor
 * @param kv_cache_strides     Strides of KV cache tensor
 * @param slot_mapping_sizes   Sizes of slot mapping tensor
 * @param slot_mapping_strides Strides of slot mapping tensor
 * @param kv_c_ndims            Number of dimensions in KV content tensor
 * @param k_pe_ndims            Number of dimensions in key position encoding
tensor
 * @param kv_cache_ndims        Number of dimensions in KV cache tensor
 * @param slot_mapping_ndims    Number of dimensions in slot mapping tensor
 * @param num_tokens           Number of tokens to process
 * @param kv_lora_rank         Rank of KV LoRA
 * @param pe_dim               Dimension of position encoding
 * @param block_size           Size of each cache block
 * @param kv_c_dtype           Data type of KV content tensor
 * @param k_pe_dtype           Data type of key position encoding tensor
 * @param kv_cache_dtype       Data type of KV cache tensor
 * @param kv_cache_type        Type of KV cache quantization
 * @param scale                Scale factor for quantization
 *
 * @note Memory Layout:
 *       - KV content tensor has shape [num_tokens, kv_lora_rank]
 *       - Key position encoding tensor has shape [num_tokens, pe_dim]
 *       - KV cache tensor has shape [num_blocks, block_size, (kv_lora_rank +
pe_dim)]
 *
 * @note Behavior:
 *       - Concatenates KV content and key position encoding
 *       - Maps tokens to their corresponding cache slots using slot_mapping
 *       - Supports optional quantization with scaling
 */
void concat_and_cache_mla_kernel(
    void* dev,                          // Device handle
    const void* kv_c,                   // KV content tensor data pointer
```

```
    const void* k_pe,                // Key position encoding tensor data pointer
    void* kv_cache,                  // KV cache tensor data pointer
    const int64_t* slot_mapping,     // Slot mapping data pointer
    const int64_t* kv_c_sizes,       // KV content tensor sizes
    const int64_t* kv_c_strides,     // KV content tensor strides
    const int64_t* k_pe_sizes,       // Key position encoding tensor sizes
    const int64_t* k_pe_strides,     // Key position encoding tensor strides
    const int64_t* kv_cache_sizes,   // KV cache tensor sizes
    const int64_t* kv_cache_strides, // KV cache tensor strides
    const int64_t* slot_mapping_sizes,  // Slot mapping tensor sizes
    const int64_t* slot_mapping_strides,// Slot mapping tensor strides
    int kv_c_ndims,                   // Number of dimensions in KV content tensor
    int k_pe_ndims,                   // Number of dimensions in key position encoding tensor
    int kv_cache_ndims,               // Number of dimensions in KV cache tensor
    int slot_mapping_ndims,           // Number of dimensions in slot mapping tensor
    int num_tokens,                   // Number of tokens to process
    int kv_lora_rank,                 // Rank of KV LoRA
    int pe_dim,                       // Dimension of position encoding
    int block_size,                   // Size of each cache block
    int kv_c_dtype,                   // Data type of KV content tensor
    int k_pe_dtype,                   // Data type of key position encoding tensor
    int kv_cache_dtype,               // Data type of KV cache tensor
    const char* kv_cache_type,        // Type of KV cache quantization
    const float* scale                // Scale factor for quantization size = [1]
);
```

ISSUE

- [ ] 支持非 contiguous 输入
- [ ] Cuda kernel 支持的数据类型：float32， float16， bfloat16，量化类型 __NV_E4M3/__NV_E5M2

_moe_C::moe_align_block_size[vLLM]

参考信息

moe_align_block_size 的核心功能是：

1. 重排序 tokens：
    a. 根据分配给它们的专家 ID 对输入 tokens 进行重排序
    b. 确保分配给同一专家的 tokens 在内存中连续存储
2. 对齐到块大小：
    a. 将每个专家处理的 tokens 数量填充到 block_size 的倍数，这种对齐优化了 GPU 上的并行计算效率
3. 生成专家映射：
    a. 为每个块分配对应的专家 ID
    b. 记录重排序后的 token 总数（包括填充）

参考实现：https://github.com/vllm-project/vllm/blob/18ed3132d2bfe1df9a74729457b69243955221e8/csrc/moe/moe_align_sum_kernels.cu#L296

Function 接口

代码块

```
/**
 * @brief Aligns tokens by expert and arranges them in blocks for efficient MoE computation
 *
 * @param dev                 Device handle for CUDA/CPU device management
 * @param topk_ids            Tensor containing expert IDs for each token [num_tokens]
 * @param sorted_token_ids    Output tensor to store sorted token indices [num_tokens_post_pad]
 * @param expert_ids          Output tensor to store expert ID for each block [num_blocks]
 * @param total_tokens_post_pad Output scalar to store total number of tokens after padding
 * @param topk_ids_sizes      Sizes of topk_ids tensor
 * @param topk_ids_strides    Strides of topk_ids tensor
 * @param sorted_token_ids_sizes Sizes of sorted_token_ids tensor
 * @param sorted_token_ids_strides Strides of sorted_token_ids tensor
 * @param expert_ids_sizes    Sizes of expert_ids tensor
 * @param expert_ids_strides  Strides of expert_ids tensor
 * @param topk_ids_ndims       Number of dimensions in topk_ids tensor
 * @param sorted_token_ids_ndims Number of dimensions in sorted_token_ids tensor
 * @param expert_ids_ndims     Number of dimensions in expert_ids tensor
 * @param num_tokens          Number of tokens to process
 * @param num_experts         Number of experts in the MoE model
 * @param block_size          Size of each processing block (for alignment)
```

```
 * @param topk_ids_dtype      Data type of topk_ids tensor
 * @param use_global_memory    Whether to use global memory instead of shared memory
 * @param use_16bit_counters   Whether to use 16-bit counters to save shared memory
 *
 * @note Memory Layout:
 *       - topk_ids: Contains expert ID assigned to each token
 *       - sorted_token_ids: Contains token indices sorted by expert ID, with padding
 *       - expert_ids: Contains expert ID for each block of tokens
 *
 * @note Behavior:
 *       - Groups tokens by expert ID and arranges them in blocks of size block_size
 *       - Pads each expert's tokens to a multiple of block_size
 *       - Returns the total number of tokens after padding
 *       - Automatically selects between shared memory and global memory implementation
 *
 */
void moe_align_block_size_kernel(
    void* dev,                          // Device handle
    const void* topk_ids,               // Expert IDs for each token
    void* sorted_token_ids,             // Output sorted token indices
    void* expert_ids,                   // Output expert ID for each block
    void* total_tokens_post_pad,        // Output total tokens after padding
    const int64_t* topk_ids_sizes,      // Sizes of topk_ids tensor
    const int64_t* topk_ids_strides,    // Strides of topk_ids tensor
    const int64_t* sorted_token_ids_sizes,    // Sizes of sorted_token_ids tensor
    const int64_t* sorted_token_ids_strides,  // Strides of sorted_token_ids tensor
    const int64_t* expert_ids_sizes,    // Sizes of expert_ids tensor
    const int64_t* expert_ids_strides,  // Strides of expert_ids tensor
    int topk_ids_ndims,                  // Number of dimensions in topk_ids tensor
    int sorted_token_ids_ndims,          // Number of dimensions in sorted_token_ids tensor
    int expert_ids_ndims,                // Number of dimensions in expert_ids tensor
    int num_tokens,                     // Number of tokens to process
    int num_experts,                    // Number of experts in the MoE model
    int block_size,                     // Size of each processing block
    int topk_ids_dtype,                 // Data type of topk_ids tensor
    bool use_global_memory,             // Whether to use global memory
    bool use_16bit_counters             // Whether to use 16-bit counters
```

```
58   );
```

ISSUE

- [ ] 对于 topk_ids，默认是 int32 类型，如果 topk_ids<65535，可选择 int16 类型

代码块

```
1    if (shared_mem_i16 < device_max_shared_mem && topk_ids.numel() <= 65535) {
2      use_i16 = true;
3    }
```

- [ ] 其余_ids 使用 int32 类型
- [ ] 支持非 contiguous

_moe_C::moe_sum[vLLM]

参考信息

moe_sum 的核心功能是：

1. 专家输出求和：
   a. 对每个 token，将其分配到的多个专家（通常是 top-k 个专家）的输出进行求和，生成最终的 token 表示
2. 高效实现：
   a. 针对常见的 top-k 值（2、3、4）提供了优化的 CUDA 实现
   b. 对于其他 top-k 值，回退到 PyTorch 的通用求和操作

参考实现：https://github.com/vllm-project/vllm/blob/18ed3132d2bfe1df9a74729457b69243955221e8/csrc/moe/moe_align_sum_kernels.cu#L422

Function 接口

代码块

```
1   /**
2    * @brief Sums outputs from multiple experts for each token in MoE models
3    *
4    * @param dev                  Device handle for CUDA/CPU device management
5    * @param out                  Output tensor to store summed results
    [num_tokens, hidden_size]
```

```
 * @param input             Input tensor with expert outputs [num_tokens, topk, hidden_size]
 * @param out_sizes         Sizes of output tensor
 * @param out_strides       Strides of output tensor
 * @param input_sizes       Sizes of input tensor
 * @param input_strides     Strides of input tensor
 * @param out_ndim          Number of dimensions in output tensor
 * @param input_ndim        Number of dimensions in input tensor
 * @param num_tokens        Number of tokens to process
 * @param topk              Number of experts per token (typically 1-4)
 * @param hidden_size       Size of the hidden dimension
 * @param out_dtype         Data type of output tensor
 * @param input_dtype       Data type of input tensor
 *
 * @note Memory Layout:
 *       - input: Contains outputs from topk experts for each token
 *       - out: Contains the sum of expert outputs for each token
 *
 * @note Behavior:
 *       - For each token, sums the outputs from its topk experts
 *       - Has optimized implementations for topk=2, 3, and 4
 *       - Falls back to PyTorch's sum for other topk values
 */
void moe_sum_kernel(
    void* dev,                          // Device handle
    void* out,                          // Output tensor data pointer
    const void* input,                  // Input tensor data pointer
    const int64_t* out_sizes,           // Output tensor sizes
    const int64_t* out_strides,         // Output tensor strides
    const int64_t* input_sizes,         // Input tensor sizes
    const int64_t* input_strides,       // Input tensor strides
    int out_ndim,                       // Number of dimensions in output tensor
    int input_ndim,                     // Number of dimensions in input tensor
    int num_tokens,                     // Number of tokens to process
    int topk,                           // Number of experts per token
    int hidden_size,                    // Size of the hidden dimension
    int out_dtype,                      // Data type of output tensor
    int input_dtype                     // Data type of input tensor
);
```

ISSUE

- [ ] Cuda 优化的 top = 2，3，4 的 kernel 中需要保证 input，output 在最后两个维度上是 contiguous 的

- [ ] 如果需要处理非连续张量，可以先调用 。contiguous（） 方法或者使用 at::sum（input， 1） 代替，但可能牺牲一些性能
- [ ] 支持的浮点类型 float32、float16、bfloat16

_vllm_fa2_C::varlen_fwd[vLLM]

查看 OP 的开发文档 中的 FA2 部分

vllm::inplace_fused_experts[vLLM]

参考信息

参考实现：https://github.com/vllm-project/vllm/blob/e92694b6fe264a85371317295bca6643508034ef/vllm/model_executor/layers/fused_moe/fused_moe.py#L1119 https://github.com/vllm-project/vllm/blob/e92694b6fe264a85371317295bca6643508034ef/vllm/model_executor/layers/fused_moe/fused_moe.py#L623C5-L623C28

这是一个组合 op，fused_experts_impl 函数调用了以下关键操作：

1. moe_align_block_size：重排序和对齐 tokens OP 的开发文档
2. invoke_fused_moe_kernel（两次）：执行高效的矩阵乘法
3. silu_and_mul：应用 SiLU 激活函数 OP 的开发文档
4. moe_sum：合并多个专家的输出 OP 的开发文档

invoke_fused_moe_kernel 在 cuda 中分为两种 kernel：**fused_moe_kernel_gptq_awq 和 fused_moe_kernel**

fused_moe_kernel_gptq_awq

1. 专为 GPTQ 和 AWQ 量化方案设计
2. 支持 INT4 （4 位整数） 和 INT8 （8 位整数） 量化的权重
3. 处理更复杂的量化参数，包括零点 （zero point）

fused_moe_kernel

1. 支持更通用的量化方案
2. 处理 FP8 （8 位浮点） 和 INT8 量化
3. 更灵活的块大小和分组配置

	fused_moe_kernel_gptq_awq	fused_moe_kernel
量化方案	INT4, INT8	FP8, INT8
零点处理	需要额外内存访问	不需要
特殊内存布局	INT4 打包存储	FP8 双向量化
分组策略	一维分组 (group_size)	二维分组 (group_n, group_k)

运行逻辑：

- 线程索引和输出位置计算
- 专家和 token 映射确定
- 权重加载和反量化
- 输入加载和处理
- 矩阵乘法计算
- 路由权重应用
- 结果存储

Function 接口

fused_moe_kernel_gptq_awq

代码块

```
/**
 * @brief Performs fused MoE computation with GPTQ/AWQ quantized weights
 *
 * @param dev                 Device handle for CUDA/CPU device management
 * @param input               Input tensor [num_tokens, hidden_size]
 * @param weights             Quantized expert weights [num_experts,
output_size, hidden_size]
 * @param output              Output tensor [num_tokens_post_pad, topk,
output_size]
 * @param weight_scales       Weight scaling factors for dequantization
 * @param weight_zero_points  Weight zero points for dequantization (optional)
 * @param topk_weights        Routing weights for each token [num_tokens,
topk]
 * @param topk_ids            Expert IDs for each token [num_tokens, topk]
 * @param sorted_token_ids    Sorted token indices [num_tokens_post_pad]
 * @param expert_ids          Expert ID for each block [num_blocks]
 * @param num_tokens_post_pad Total number of tokens after padding
 * @param input_sizes         Sizes of input tensor
 * @param input_strides       Strides of input tensor
 * @param weights_sizes       Sizes of weights tensor
 * @param weights_strides     Strides of weights tensor
```

```
 19   * @param output_sizes       Sizes of output tensor
 20   * @param output_strides     Strides of output tensor
 21   * @param weight_scales_sizes  Sizes of weight_scales tensor
 22   * @param weight_scales_strides Strides of weight_scales tensor
 23   * @param weight_zero_points_sizes Sizes of weight_zero_points tensor (if not
      null)
 24   * @param weight_zero_points_strides Strides of weight_zero_points tensor (if
      not null)
 25   * @param topk_weights_sizes   Sizes of topk_weights tensor
 26   * @param topk_weights_strides Strides of topk_weights tensor
 27   * @param topk_ids_sizes       Sizes of topk_ids tensor
 28   * @param topk_ids_strides     Strides of topk_ids tensor
 29   * @param sorted_token_ids_sizes Sizes of sorted_token_ids tensor
 30   * @param sorted_token_ids_strides Strides of sorted_token_ids tensor
 31   * @param expert_ids_sizes     Sizes of expert_ids tensor
 32   * @param expert_ids_strides   Strides of expert_ids tensor
 33   * @param input_ndim           Number of dimensions in input tensor
 34   * @param weights_ndim         Number of dimensions in weights tensor
 35   * @param output_ndim          Number of dimensions in output tensor
 36   * @param weight_scales_ndim   Number of dimensions in weight_scales tensor
 37   * @param weight_zero_points_ndim Number of dimensions in weight_zero_points
      tensor
 38   * @param topk_weights_ndim    Number of dimensions in topk_weights tensor
 39   * @param topk_ids_ndim        Number of dimensions in topk_ids tensor
 40   * @param sorted_token_ids_ndim Number of dimensions in sorted_token_ids tensor
 41   * @param expert_ids_ndim      Number of dimensions in expert_ids tensor
 42   * @param num_tokens           Number of tokens to process
 43   * @param num_experts          Number of experts in the model
 44   * @param hidden_size          Size of input hidden dimension
 45   * @param output_size          Size of output dimension
 46   * @param topk                 Number of experts per token
 47   * @param block_size_m         Block size for M dimension
 48   * @param block_size_n         Block size for N dimension
 49   * @param block_size_k         Block size for K dimension
 50   * @param group_size_m         Group size for M dimension
 51   * @param group_size           Quantization group size
 52   * @param mul_routed_weight    Whether to multiply by routing weights
 53   * @param input_dtype          Data type of input tensor
 54   * @param weights_dtype        Data type of weights tensor
 55   * @param output_dtype         Data type of output tensor
 56   * @param weight_scales_dtype  Data type of weight_scales tensor
 57   * @param weight_zero_points_dtype Data type of weight_zero_points tensor
 58   * @param use_int4_w4a16       Whether to use INT4 weights with FP16
      activations
 59   * @param use_int8_w8a16       Whether to use INT8 weights with FP16
      activations
 60   *
```

```
 61   * @note Memory Layout:
 62   *       - input: Contains token hidden states
 63   *       - weights: Contains quantized expert weights
 64   *       - output: Contains expert outputs for each token
 65   *
 66   * @note Quantization:
 67   *       - For INT4: Two 4-bit values are packed into one byte
 68   *       - For INT8: Each weight is stored as a single byte
 69   *       - Dequantization: (quant_value - zero_point) * scale
 70   *
 71   * @note Behavior:
 72   *       - Performs matrix multiplication between token hidden states and
      expert weights
 73   *       - Handles quantized weights with dequantization
 74   *       - Optionally multiplies results by routing weights
 75   *       - Outputs are arranged by token and expert
 76   */
 77  void fused_moe_kernel_gptq_awq(
 78      void* dev,                          // Device handle
 79      const void* input,                  // Input tensor data pointer
 80      const void* weights,                // Weights tensor data pointer
 81      void* output,                       // Output tensor data pointer
 82      const void* weight_scales,          // Weight scales data pointer
 83      const void* weight_zero_points,     // Weight zero points data pointer
      (can be null)
 84      const void* topk_weights,           // Topk weights data pointer
 85      const void* topk_ids,               // Topk IDs data pointer
 86      const void* sorted_token_ids,       // Sorted token IDs data pointer
 87      const void* expert_ids,             // Expert IDs data pointer
 88      const void* num_tokens_post_pad,    // Number of tokens post padding
 89      const int64_t* input_sizes,         // Input tensor sizes
 90      const int64_t* input_strides,       // Input tensor strides
 91      const int64_t* weights_sizes,       // Weights tensor sizes
 92      const int64_t* weights_strides,     // Weights tensor strides
 93      const int64_t* output_sizes,        // Output tensor sizes
 94      const int64_t* output_strides,      // Output tensor strides
 95      const int64_t* weight_scales_sizes, // Weight scales tensor sizes
 96      const int64_t* weight_scales_strides, // Weight scales tensor strides
 97      const int64_t* weight_zero_points_sizes, // Weight zero points tensor
      sizes (if not null)
 98      const int64_t* weight_zero_points_strides, // Weight zero points tensor
      strides (if not null)
 99      const int64_t* topk_weights_sizes,  // Topk weights tensor sizes
100      const int64_t* topk_weights_strides, // Topk weights tensor strides
101      const int64_t* topk_ids_sizes,      // Topk IDs tensor sizes
102      const int64_t* topk_ids_strides,    // Topk IDs tensor strides
103      const int64_t* sorted_token_ids_sizes, // Sorted token IDs tensor sizes
```

```
    const int64_t* sorted_token_ids_strides, // Sorted token IDs tensor strides
    const int64_t* expert_ids_sizes,    // Expert IDs tensor sizes
    const int64_t* expert_ids_strides,  // Expert IDs tensor strides
    int input_ndim,                     // Number of dimensions in input tensor
    int weights_ndim,                   // Number of dimensions in weights
tensor
    int output_ndim,                    // Number of dimensions in output
tensor
    int weight_scales_ndim,             // Number of dimensions in weight
scales tensor
    int weight_zero_points_ndim,        // Number of dimensions in weight zero
points tensor
    int topk_weights_ndim,              // Number of dimensions in topk
weights tensor
    int topk_ids_ndim,                  // Number of dimensions in topk IDs
tensor
    int sorted_token_ids_ndim,          // Number of dimensions in sorted
token IDs tensor
    int expert_ids_ndim,                // Number of dimensions in expert IDs
tensor
    int num_tokens,                     // Number of tokens to process
    int num_experts,                    // Number of experts in the model
    int hidden_size,                    // Size of input hidden dimension
    int output_size,                    // Size of output dimension
    int topk,                           // Number of experts per token
    int block_size_m,                   // Block size for M dimension
    int block_size_n,                   // Block size for N dimension
    int block_size_k,                   // Block size for K dimension
    int group_size_m,                   // Group size for M dimension
    int group_size,                     // Quantization group size
    bool mul_routed_weight,             // Whether to multiply by routing
weights
    int input_dtype,                    // Data type of input tensor
    int weights_dtype,                  // Data type of weights tensor
    int output_dtype,                   // Data type of output tensor
    int weight_scales_dtype,            // Data type of weight scales tensor
    int weight_zero_points_dtype,       // Data type of weight zero points
tensor
    bool use_int4_w4a16,                // Whether to use INT4 weights with
FP16 activations
    bool use_int8_w8a16                 // Whether to use INT8 weights with
FP16 activations
);
```

fused_moe_kernel

```
/**
 * @brief Performs fused MoE computation with FP8/INT8 quantized weights
 *
 * @param dev                  Device handle for CUDA/CPU device management
 * @param input                Input tensor [num_tokens, hidden_size]
 * @param weights              Expert weights [num_experts, output_size, hidden_size]
 * @param output               Output tensor [num_tokens_post_pad, topk, output_size]
 * @param input_scales         Input scaling factors for FP8 quantization
 * @param weight_scales        Weight scaling factors for quantization
 * @param topk_weights         Routing weights for each token [num_tokens, topk]
 * @param topk_ids             Expert IDs for each token [num_tokens, topk]
 * @param sorted_token_ids     Sorted token indices [num_tokens_post_pad]
 * @param expert_ids           Expert ID for each block [num_blocks]
 * @param num_tokens_post_pad  Total number of tokens after padding
 * @param input_sizes          Sizes of input tensor
 * @param input_strides        Strides of input tensor
 * @param weights_sizes        Sizes of weights tensor
 * @param weights_strides      Strides of weights tensor
 * @param output_sizes         Sizes of output tensor
 * @param output_strides       Strides of output tensor
 * @param input_scales_sizes   Sizes of input_scales tensor
 * @param input_scales_strides Strides of input_scales tensor
 * @param weight_scales_sizes  Sizes of weight_scales tensor
 * @param weight_scales_strides Strides of weight_scales tensor
 * @param topk_weights_sizes   Sizes of topk_weights tensor
 * @param topk_weights_strides Strides of topk_weights tensor
 * @param topk_ids_sizes       Sizes of topk_ids tensor
 * @param topk_ids_strides     Strides of topk_ids tensor
 * @param sorted_token_ids_sizes Sizes of sorted_token_ids tensor
 * @param sorted_token_ids_strides Strides of sorted_token_ids tensor
 * @param expert_ids_sizes     Sizes of expert_ids tensor
 * @param expert_ids_strides   Strides of expert_ids tensor
 * @param input_ndim           Number of dimensions in input tensor
 * @param weights_ndim         Number of dimensions in weights tensor
 * @param output_ndim          Number of dimensions in output tensor
 * @param input_scales_ndim    Number of dimensions in input_scales tensor
 * @param weight_scales_ndim   Number of dimensions in weight_scales tensor
 * @param topk_weights_ndim    Number of dimensions in topk_weights tensor
 * @param topk_ids_ndim        Number of dimensions in topk_ids tensor
 * @param sorted_token_ids_ndim Number of dimensions in sorted_token_ids tensor
 * @param expert_ids_ndim      Number of dimensions in expert_ids tensor
 * @param num_tokens           Number of tokens to process
 * @param num_experts          Number of experts in the model
 * @param hidden_size          Size of input hidden dimension
```

```
 * @param output_size        Size of output dimension
 * @param topk               Number of experts per token
 * @param block_size_m       Block size for M dimension
 * @param block_size_n       Block size for N dimension
 * @param block_size_k       Block size for K dimension
 * @param group_size_m       Group size for M dimension
 * @param group_n            Group size for N dimension in block quantization
 * @param group_k            Group size for K dimension in block quantization
 * @param mul_routed_weight  Whether to multiply by routing weights
 * @param input_dtype        Data type of input tensor
 * @param weights_dtype      Data type of weights tensor
 * @param output_dtype       Data type of output tensor
 * @param input_scales_dtype Data type of input_scales tensor
 * @param weight_scales_dtype Data type of weight_scales tensor
 * @param use_fp8_w8a8       Whether to use FP8 weights with FP8 activations
 * @param use_int8_w8a16     Whether to use INT8 weights with FP16 activations
 *
 * @note Memory Layout:
 *       - input: Contains token hidden states
 *       - weights: Contains expert weights
 *       - output: Contains expert outputs for each token
 *
 * @note Quantization:
 *       - For FP8: Both weights and activations use 8-bit floating point
 *       - For INT8: Weights use 8-bit integers, activations use 16-bit floating point
 *       - Block quantization: Different scale factors for different blocks
 *
 * @note Behavior:
 *       - Performs matrix multiplication between token hidden states and expert weights
 *       - Handles quantized weights and activations
 *       - Optionally multiplies results by routing weights
 *       - Outputs are arranged by token and expert
 */
void fused_moe_kernel(
    void* dev,                          // Device handle
    const void* input,                  // Input tensor data pointer
    const void* weights,                // Weights tensor data pointer
    void* output,                       // Output tensor data pointer
    const void* input_scales,           // Input scales data pointer
    const void* weight_scales,          // Weight scales data pointer
    const void* topk_weights,           // Topk weights data pointer
    const void* topk_ids,               // Topk IDs data pointer
    const void* sorted_token_ids,       // Sorted token IDs data pointer
    const void* expert_ids,             // Expert IDs data pointer
```

```
89      const void* num_tokens_post_pad,     // Number of tokens post padding
90      const int64_t* input_sizes,          // Input tensor sizes
91      const int64_t* input_strides,        // Input tensor strides
92      const int64_t* weights_sizes,        // Weights tensor sizes
93      const int64_t* weights_strides,      // Weights tensor strides
94      const int64_t* output_sizes,         // Output tensor sizes
95      const int64_t* output_strides,       // Output tensor strides
96      const int64_t* input_scales_sizes,   // Input scales tensor sizes
97      const int64_t* input_scales_strides, // Input scales tensor strides
98      const int64_t* weight_scales_sizes, // Weight scales tensor sizes
99      const int64_t* weight_scales_strides, // Weight scales tensor strides
100     const int64_t* topk_weights_sizes,  // Topk weights tensor sizes
101     const int64_t* topk_weights_strides, // Topk weights tensor strides
102     const int64_t* topk_ids_sizes,       // Topk IDs tensor sizes
103     const int64_t* topk_ids_strides,     // Topk IDs tensor strides
104     const int64_t* sorted_token_ids_sizes, // Sorted token IDs tensor sizes
105     const int64_t* sorted_token_ids_strides, // Sorted token IDs tensor strides
106     const int64_t* expert_ids_sizes,     // Expert IDs tensor sizes
107     const int64_t* expert_ids_strides,   // Expert IDs tensor strides
108     int input_ndim,                      // Number of dimensions in input tensor
109     int weights_ndim,                    // Number of dimensions in weights
    tensor
110     int output_ndim,                     // Number of dimensions in output
    tensor
111     int input_scales_ndim,               // Number of dimensions in input
    scales tensor
112     int weight_scales_ndim,              // Number of dimensions in weight
    scales tensor
113     int topk_weights_ndim,               // Number of dimensions in topk
    weights tensor
114     int topk_ids_ndim,                   // Number of dimensions in topk IDs
    tensor
115     int sorted_token_ids_ndim,           // Number of dimensions in sorted
    token IDs tensor
116     int expert_ids_ndim,                 // Number of dimensions in expert IDs
    tensor
117     int num_tokens,                      // Number of tokens to process
118     int num_experts,                     // Number of experts in the model
119     int hidden_size,                     // Size of input hidden dimension
120     int output_size,                     // Size of output dimension
121     int topk,                            // Number of experts per token
122     int block_size_m,                    // Block size for M dimension
123     int block_size_n,                    // Block size for N dimension
124     int block_size_k,                    // Block size for K dimension
125     int group_size_m,                    // Group size for M dimension
126     int group_n,                         // Group size for N dimension in block
    quantization
```

```
    int group_k,                    // Group size for K dimension in block quantization
    bool mul_routed_weight,         // Whether to multiply by routing weights
    int input_dtype,                // Data type of input tensor
    int weights_dtype,              // Data type of weights tensor
    int output_dtype,               // Data type of output tensor
    int input_scales_dtype,         // Data type of input scales tensor
    int weight_scales_dtype,        // Data type of weight scales tensor
    bool use_fp8_w8a8,              // Whether to use FP8 weights with FP8 activations
    bool use_int8_w8a16             // Whether to use INT8 weights with FP16 activations
);
```

ISSUE

- [ ] 两个内核都支持张量非contiguous
- [ ] 支持的数据类型：

	fused_moe_kernel_gptq_awq	fused_moe_kernel
输入数据类型	FP32, FP16, BF16	FP32, FP16, BF16, UINT8 (FP8)
权重数据类型	INT8, 打包 INT4	FP32, FP16, BF16, INT8, UINT8 (FP8)
计算精度	FP32 或 FP16	FP32 或 FP16

vllm::unified_attention[vLLM]

参考信息

源代码：https://github.com/vllm-project/vllm/blob/e92694b6fe264a85371317295bca6643508034ef/vllm/attention/layer.py#L297

这是一个用来调用attention layer forward()方法的统一接口

代码块

```
def unified_attention(
    query: torch.Tensor,
    key: torch.Tensor,
    value: torch.Tensor,
    layer_name: str,
) -> torch.Tensor:
    forward_context: ForwardContext = get_forward_context()
    attn_metadata = forward_context.attn_metadata
    self = forward_context.attn_layers[layer_name]
```

```
        kv_cache = self.kv_cache[forward_context.virtual_engine]
        return self.impl.forward(self, query, key, value, kv_cache, attn_metadata)
```

不涉及到kernel的开发，起到一个分发到具体attention实现的作用，建议此处使用一个更加高级的API，在分发至不同attention kernel的时候再针对具体attention 需要的参数开发对应C++接口内容，比如说这类接口 OP 的开发文档

Function 接口

代码块

```
/**
 * @brief Attention metadata structure containing all necessary information for attention computation
 */
struct AttentionMetadata {
    // Sequence information
    torch::Tensor block_tables;        // [num_seqs, max_blocks_per_seq]
    torch::Tensor context_lens;        // [num_seqs]
    torch::Tensor max_context_len;     // scalar
    torch::Tensor seq_lens;            // [num_seqs]
    torch::Tensor seq_start_loc;       // [num_seqs]

    // Batch information
    int num_tokens;                    // Total number of tokens
    int num_seqs;                      // Number of sequences
    int max_seq_len;                   // Maximum sequence length

    // Attention mechanism
    torch::Tensor alibi_slopes;        // [num_heads] or empty
    torch::Tensor attn_bias;           // Optional attention bias
    torch::Tensor slot_mapping;        // [num_tokens]
    int sliding_window_size;           // Size of sliding window (-1 if not used)

    // Quantization
    bool enable_kv_scale_calc;         // Flag to enable KV scale calculation

    // Prefill/decode information
    int num_prefills;                  // Number of prefill requests
    int num_prefill_tokens;            // Number of prefill tokens
    int num_decode_tokens;             // Number of decode tokens
};

/**
 * @brief KV Cache structure containing key and value cache tensors
```

```
 */
struct KVCache {
    torch::Tensor key_cache;          // [num_blocks, block_size, num_kv_heads, head_size]
    torch::Tensor value_cache;        // [num_blocks, block_size, num_kv_heads, head_size]
    int block_size;                   // Size of each block
    int max_blocks;                   // Maximum number of blocks
};

/**
 * @brief Performs attention computation with support for various attention mechanisms
 *
 * @param output               Output tensor to store attention results [batch_size, seq_len, hidden_size]
 * @param query                Query tensor [batch_size, seq_len, hidden_size]
 * @param key                  Key tensor [batch_size, seq_len, hidden_size]
 * @param value                Value tensor [batch_size, seq_len, hidden_size]
 * @param kv_cache             KV cache structure containing key and value cache tensors
 * @param metadata             Attention metadata structure
 * @param num_heads            Number of attention heads
 * @param num_kv_heads         Number of key/value heads (for MQA/GQA)
 * @param head_size            Size of each attention head
 * @param scale                Scaling factor for attention scores
 * @param k_scale              Scale factor for key (for FP8 quantization)
 * @param v_scale              Scale factor for value (for FP8 quantization)
 * @param attn_type            Type of attention (0=decoder, 1=encoder, 2=encoder_only, 3=encoder_decoder)
 *
 * @return torch::Tensor       Output tensor with attention results
 *
 * @note This is a higher-level interface that uses PyTorch tensors directly
 * @note It internally calls the lower-level attention_forward function
 */
torch::Tensor attention_forward(
    torch::Tensor query,              // Query tensor
    torch::Tensor key,                // Key tensor
    torch::Tensor value,              // Value tensor
    const KVCache& kv_cache,          // KV cache structure
    const AttentionMetadata& metadata, // Attention metadata
    int num_heads,                    // Number of attention heads
    int num_kv_heads,                 // Number of key/value heads
    int head_size,                    // Size of each attention head
    float scale,                      // Scaling factor for attention scores
    float k_scale,                    // Scale factor for key (for FP8)
```

```
    float v_scale,                  // Scale factor for value (for FP8)
    int attn_type,                  // Type of attention
    torch::optional<torch::Tensor> output = torch::nullopt // Optional pre-allocated output tensor
);
```

ISSUE

不需要kernel

vllm::unified_attention_with_output[vLLM]

参考信息

https://github.com/vllm-project/vllm/blob/6e1fc61f0fb90c37f0d4a1a8f76235a6e4e1103c/vllm/attention/layer.py#L328

这是一个用来调用attention layer forward()方法的统一接口，与上面的相比只有一点不同，参数中包含output，可以参考上一个中的高级API

代码块

```
def unified_attention_with_output(
    query: torch.Tensor,
    key: torch.Tensor,
    value: torch.Tensor,
    output: torch.Tensor,
    layer_name: str,
) -> None:
    forward_context: ForwardContext = get_forward_context()
    attn_metadata = forward_context.attn_metadata
    self = forward_context.attn_layers[layer_name]
    kv_cache = self.kv_cache[forward_context.virtual_engine]
    self.impl.forward(self,
                      query,
                      key,
                      value,
                      kv_cache,
                      attn_metadata,
                      output=output)
```

__scaled_mm

参考信息

NV 文档：https://docs.nvidia.com/cuda/cublas/#id12

使用场景参考：https://dev-discuss.pytorch.org/t/float8-in-pytorch-1-x/1815

Pytorch 源码：算子 C++地址：https://github.com/pytorch/pytorch/blob/0b0d28accdcabfeb3e60cac536749d86138222f8/aten/src/ATen/native/cuda/Blas.cpp#L1365，cuBlasLt Kernel 调用地址：https://github.com/pytorch/pytorch/blob/0b0d28accdcabfeb3e60cac536749d86138222f8/aten/src/ATen/cuda/CUDABlas.cpp#L1513

我们主要需要支持的格式：kFloat8_e5m2， kFloat8_e4m3fn

需要支持两种 scale：tensor wise&row rise

CUDA Kernel 对 Tensor 要求来源：https://docs.nvidia.com/cuda/cublas/#id83

对于 tensor wise scaling 的情况，参考代码

前期的支持需求：

output_fp32 = A_fp8@B_fp8*a_scale_fp32*b_scale_fp32

out dtype: float32

scale a dtype: float32

scale b dtype: float32

scale res: 初期不需要支持

input a dtype: float8_e4m3fn

input b dtype: float8_e4m3fn

bias: 初期不需要支持

代码块

```
import os
import torch

a = torch.eye(16, dtype=torch.float32)
a[0, 1] = 2
a[1, 0] = 3
a[1, 1] = 4
b = torch.eye(16, dtype=torch.float32)
b[0, 1] = 2
b[1, 0] = 3
b[1, 1] = 4
res_scale = 1.
a = a.cuda() / res_scale
b = b.cuda() / res_scale
```

```
15
16  c = a.mm(b)[:2,:2]
17
18  """
19  tensor([[ 7., 10.],
20          [15., 22.]], device='cuda:0')
21  """
22  print(c)
23
24  a_scale = a.abs().max()
25  b_scale = b.abs().max()
26
27  a_fp8 = (a / a_scale).to(dtype=torch.float8_e4m3fn)
28  b_fp8 = (b / b_scale).to(dtype=torch.float8_e4m3fn)
29
30  # to column-major
31  b_fp8 = b_fp8.t().contiguous().t()
32
33  c2 = torch._scaled_mm(a_fp8, b_fp8, a_scale, b_scale)[:2,:2]
34  """
35  tensor([[ 7., 10.],
36          [15., 22.]], device='cuda:0', dtype=torch.float8_e4m3fn)
37  """
38  print(c2)
39
40  c3 = torch._scaled_mm(a_fp8, b_fp8, a_scale, b_scale, out_dtype=torch.float32)
    [:2,:2]
41  """
42  tensor([[ 7., 10.],
43          [15., 22.]], device='cuda:0')
44  """
45  print(c3)
46
47
48  scale_res = a_scale * b_scale
49
50  c4 = torch._scaled_mm(a_fp8, b_fp8, torch.tensor([1.0]).cuda(),
    torch.tensor([1.0]).cuda(), scale_result=scale_res)[:2,:2]
51  """
52  tensor([[ 7., 10.],
53          [15., 22.]], device='cuda:0', dtype=torch.float8_e4m3fn)
54  """
55  print(c4)
56
57  bias = torch.tensor([1.0] * 16, dtype=torch.bfloat16).cuda()
58  c5 = torch._scaled_mm(a_fp8, b_fp8, a_scale, b_scale,
    out_dtype=torch.float8_e4m3fn, bias=bias)[:2,:2]
```

```
"""
tensor([[ 8., 11.],
        [16., 24.]], device='cuda:0', dtype=torch.float8_e4m3fn)
"""
print(c5)


```

Tensorwide scaling is enabled when `CUBLASLT_MATMUL_DESC_X_SCALE_MODE` attributes (here `X` stands for `A`, `B`, `C`, `D`, or `AUX`; see cublasLtMatmulDescAttributes_t) for all FP8-precision tensors is set to `CUBLASLT_MATMUL_MATRIX_SCALE_SCALAR_32F` (this is the default value for FP8 tensors). In such case, the matmul operation in cuBLAS is defined in the following way (assuming, for exposition, that all tensors are using an FP8 precision):

$$D = scale_D \cdot (\alpha \cdot scale_A \cdot scale_B \cdot \mathrm{op}(A)\mathrm{op}(B) + \beta \cdot scale_C \cdot C).$$

Here A, B, and C are input tensors, and $scale_A$, $scale_B$, $scale_C$, $scale_D$, α, and β are input scalars. This differs from the other matrix multiplication routines because of this addition of scaling factors for each matrix. The $scale_A$, $scale_B$, and $scale_C$ are used for de-quantization, and $scale_D$ is used for quantization. Note that all the scaling factors are applied multiplicatively. This means that sometimes it is necessary to use a scaling factor or its reciprocal depending on the context in which it is applied. For more information on FP8, see cublasLtMatmul() and cublasLtMatmulDescAttributes_t.

For such matrix multiplications, epilogues and the absolute maximums of intermediate values are computed as follows:

$$\begin{aligned} Aux_{temp} &= \alpha \cdot scale_A \cdot scale_B \cdot \mathrm{op}(A)\mathrm{op}(B) + \beta \cdot scale_C \cdot C, \\ D_{temp} &= Epilogue(Aux_{temp}), \\ amax_D &= absmax(D_{temp}), \\ amax_{Aux} &= absmax(Aux_{temp}), \\ D &= scale_D * D_{temp}, \\ Aux &= scale_{Aux} * Aux_{temp}. \end{aligned}$$

Here Aux is an auxiliary output of matmul consisting of the values that are passed to an epilogue function like GELU, $scale_{Aux}$ is an optional scaling factor that can be applied to Aux, and $amax_{Aux}$ is the maximum absolute value in Aux before scaling. For more information, see attributes `CUBLASLT_MATMUL_DESC_AMAX_D_POINTER` and `CUBLASLT_MATMUL_DESC_EPILOGUE_AUX_AMAX_POINTER` in cublasLtMatmulDescAttributes_t.

See the table below when using FP8 kernels:

Table 3. When A, B, C, and D Use Layouts for FP8

AType	BType	CType	DType	Bias Type
CUDA_R_8F_E4M3	CUDA_R_8F_E4M3	CUDA_R_16BF	CUDA_R_16BF	CUDA_R_16BF [8]
			CUDA_R_8F_E4M3	CUDA_R_16BF [8]
		CUDA_R_16F	CUDA_R_16F	CUDA_R_16F [8]
			CUDA_R_8F_E4M3	CUDA_R_16F [8]
		CUDA_R_32F	CUDA_R_32F	CUDA_R_16BF [8]
	CUDA_R_8F_E5M2	CUDA_R_16BF	CUDA_R_16BF	CUDA_R_16BF [8]
			CUDA_R_8F_E4M3	CUDA_R_16BF [8]
			CUDA_R_8F_E5M2	CUDA_R_16BF [8]
		CUDA_R_16F	CUDA_R_16F	CUDA_R_16F [8]
			CUDA_R_8F_E4M3	CUDA_R_16F [8]
			CUDA_R_8F_E5M2	CUDA_R_16F [8]
		CUDA_R_32F	CUDA_R_32F	CUDA_R_16BF [8]
CUDA_R_8F_E5M2	CUDA_R_8F_E4M3	CUDA_R_16BF	CUDA_R_16BF	CUDA_R_16BF [8]
			CUDA_R_8F_E4M3	CUDA_R_16BF [8]
			CUDA_R_8F_E5M2	CUDA_R_16BF [8]
		CUDA_R_16F	CUDA_R_16F	CUDA_R_16F [8]
			CUDA_R_8F_E4M3	CUDA_R_16F [8]
			CUDA_R_8F_E5M2	CUDA_R_16F [8]
		CUDA_R_32F	CUDA_R_32F	CUDA_R_16BF [8]

CUBLASLT_EPILOGUE_BIAS = 4	Apply (broadcast) bias from the bias vector. Bias vector length must match matrix D rows, and it must be packed (such as stride between vector elements is 1). Bias vector is broadcast to all columns and added before applying the final postprocessing.

mm MXFormat

参考信息

实现方案：https://github.com/pytorch/ao/tree/main/torchao/prototype/mx_formats

NV 文档：https://docs.nvidia.com/cuda/cublas/#id12

1. MXTensor 继承 Tensor，通过__torch_dispatch__来处理：

   对于 mm

   a. Tensor @ MXTensor -> convert(Tensor->MX) @ check_and_transform_shape([MX,MX]) -> return convert(MX -> Tensor)

   b. MX @ MX -> check_and_transform_shape([MX,MX]) -> return MX

   注意：

   a. convert（Tensor->MX）时，软件需要选择 MXFormat 对应的正确 shape，并且 padding，详情参考：SiPU 1.0 Tiled Format 架构设计方案 v1.0

   b. check_and_transform_shape 主要是因为有可能 MXTensor 是由一个 MXTensor B 矩/A 矩阵转置而来，需要 check，但是由于这一步 transform 开销较大，最好有 warning

   c. B 矩阵只能采用 M32 的 tile shape

2. 更改 StorageImpl，参考 pytorch_ascend 项目，添加 desc 项，记录实际的内存 alignment 情况，实际内存大小等信息

ISSUE:

- [ ] 对于 Op 实现，是直接注册到 aten::mm 上还是创建一个新的算子
  - [ ] 直接注册到 aten::mm 上不确定这个 op 对应的所有 dispatch key 是否能正确处理 MXTensor
  - [ ] 自定义实现的话 AutoGrad 的实现还需要特别处理

mx_to_high_precision/high_precision_to_mx

mx_ptr 的 size 可以根据 tileformat 文档计算出来

- data size:
  - mxint8: M * N
  - mxfp6: M * N * 3 / 4
- Header size:
  - mxint8: M * N / 16
  - mxfp6: M * N / 32
- e.g. mxfp6, supertile(M=32, N=128*4=512), data size=12288B, header size=512B, total=12800B
- e.g. mxint8, supertile(M=32, N=32*4=128), data size=4096B, header size=256B, total=4352B

目前支持的版本为 tile to tile，框架会在上层做 linear to tile/tile to linear

```
代码块
```

```
// MX_T: sifmt::mxint8, sifmt::mxfp6e3m2, sifmt::mxfp6e2m3
// HP_T: sifmt::bfloat16, sifmt::float16, sifmt::float32
template<typename MX_T, typename HP_T>
void mx_to_high_precision(void* mx_ptr, HP_T* hp_ptr, int batch_size, int dim0, int dim1)

template<typename MX_T, typename HP_T>
void high_precision_to_mx(void* mx_ptr, HP_T* hp_ptr, int batch_size, int dim0, int dim1)
```

_empty_affine_quantized

示例：

代码块

```
# PYTHON例子
import torch

# 创建一个空的QTensor
a = torch._empty_affine_quantized(
    size=(3, 4),
    dtype=torch.quint8,
    scale=0.02,
    zero_point=64,
    device='cpu'
)

print(a)

"""
>>> tensor([[ 3.5200,  2.1400,  3.7200, -0.4200],
        [-0.4000,  1.0600, -1.2800, -1.2800],
        [-0.8000,  3.3200,  3.3200, -0.5400]], size=(3, 4), dtype=torch.quint8,
      quantization_scheme=torch.per_tensor_affine, scale=0.02, zero_point=64)
"""
```

【NOTE】不需要算子实现

参考实现

Source file: pytorch/aten/src/ATen/native/quantized/TensorFactories.cpp: 15

```cpp
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ empty ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// We explicitly pass in scale and zero_point because we don't have the infra
// ready to support quantizer in python frontend, once that is ready, we'll
// change to use quantizer
Tensor empty_affine_quantized(
    IntArrayRef size,
    std::optional<ScalarType> dtype,
    std::optional<Layout> layout,
    std::optional<Device> device,
    std::optional<bool> pin_memory,
    double scale,
    int64_t zero_point,
    std::optional<c10::MemoryFormat> optional_memory_format) {
  // See [Note: hacky wrapper removal for TensorOptions]
  TensorOptions options_ = TensorOptions().dtype(dtype).layout(layout).device(device).pinned_memory(pinned_memory: pin_memory);

  TORCH_CHECK(
    !(options_.has_memory_format() && optional_memory_format.has_value()),
    args: "Cannot set memory_format both in TensorOptions and explicit argument; please delete "
    "the redundant setter.");
  auto options: TensorOptions = options_.merge_memory_format(optional_memory_format);
  TORCH_CHECK(
      options.has_dtype(),
      args: "Must provide data type for Tensor creation functions.");
  return new_qtensor(       You, 3周前 • Uncommitted changes
      sizes: size,
      options,
      quantizer: make_per_tensor_affine_quantizer(
          scale, zero_point, scalar_type: typeMetaToScalarType(dtype: options.dtype())));
}
```

empty_affine_quantized 是_empty_affine_quantized的内部实现接口，这里调用了**new_qtensor**，创建一个空QTensor，QTensor里保存着创建的Tensor以及一个Quantizer。

Source file: pytorch/aten/src/ATen/quantized/Quantizer.cpp: 108

```
inline Tensor new_qtensor(      You, 3周前 • Uncommitted changes
    IntArrayRef sizes,
    const TensorOptions& options,
    QuantizerPtr quantizer) {
  auto memory_format: MemoryFormat = options.memory_format_opt().value_or(u: MemoryFormat::Contiguous);
  auto device: Device = options.device();
  at::Allocator* allocator = nullptr;
  // TODO: why isn't this just using GetAllocator
  if (device.is_cuda()) {
    allocator = at::detail::getCUDAHooks().getCUDADeviceAllocator();
  } else if (device.is_cpu()) {
    allocator = at::getCPUAllocator();
  } else if (device.is_meta()) {
    allocator = GetAllocator(t: kMeta);
  } else if (device.is_privateuseone()) {
    allocator = GetAllocator(t: kPrivateUse1);
  } else {
    TORCH_INTERNAL_ASSERT(0, "unrecognized device for new_qtensor: ", device);
  }

#ifdef USE_PYTORCH_QNNPACK
  if (at::globalContext().qEngine() == at::QEngine::QNNPACK) {
    TORCH_CHECK(!device.is_cuda(), "It looks like you are trying to quantize a CUDA tensor ",
                "while QNNPACK backend is enabled. Although not expected to happen in ",
                "practice, you might have done it for testing purposes. ",
                "Please, either change the quantization engine or move the tensor to a CPU.");
    allocator = c10::GetDefaultMobileCPUAllocator();
  }
#endif

  at::DispatchKey tensorDispatchKey = options.computeDispatchKey();
  native::check_size_nonnegative(size: sizes);
  auto dtype: caffe2::TypeMeta = options.dtype();
  TORCH_CHECK(
      isQIntType(t: typeMetaToScalarType(dtype)),
      "ScalarType ",
      typeMetaToScalarType(dtype),
```

```
#ifdef USE_PYTORCH_QNNPACK
  if (at::globalContext().qEngine() == at::QEngine::QNNPACK) {
    TORCH_CHECK(!device.is_cuda(), "It looks like you are trying to quantize a CUDA tensor ",
                "while QNNPACK backend is enabled. Although not expected to happen in ",
                "practice, you might have done it for testing purposes. ",
                "Please, either change the quantization engine or move the tensor to a CPU.");
    allocator = c10::GetDefaultMobileCPUAllocator();
  }
#endif

  at::DispatchKey tensorDispatchKey = options.computeDispatchKey();
  native::check_size_nonnegative(size: sizes);
  auto dtype: caffe2::TypeMeta = options.dtype();
  TORCH_CHECK(
      isQIntType(t: typeMetaToScalarType(dtype)),
      "ScalarType ",
      typeMetaToScalarType(dtype),
      " is not supported in new_qtensor.");
  auto scalar_type: ScalarType = typeMetaToScalarType(dtype);
  int64_t size_bytes = get_sub_byte_tensor_size(sizes, dtype_itemsize: dtype.itemsize(), t: scalar_type);

  auto storage = make_storage_impl(
      use_byte_size: StorageImpl::use_byte_size_t(),
      size_bytes,
      data_ptr: allocator->allocate(n: size_bytes),
      allocator,
      /*resizable=*/true,
      device_opt: device);
  auto tensor: Tensor = detail::make_tensor<QTensorImpl>(
      storage, at::DispatchKeySet(k: tensorDispatchKey), dtype, quantizer);
  get_qtensorimpl(self: tensor)->set_sizes_contiguous(new_size: sizes);
  get_qtensorimpl(self: tensor)->empty_tensor_restride(memory_format);
  return tensor;
}
```

is_neg

判断一个Tensor的DispatchKeySet中的Negative位是否被设置。

示例

代码块

```
import torch

a = torch.tensor((3, 4))
# 此时Negative bit没有被设置
print(torch.is_neg(a))

"""
>>> False
"""

# 设置一下Negative bit
b = torch._C._set_neg(b, True)
print(torch.is_neg(a))

```

```
15  """
16  >>> True
17  """
```

源码

Source file: pytorch/aten/src/ATen/core/TensorBase.h: 392

```
inline bool is_neg() const {
  return impl_->is_neg();
}
```

Source file: pytorch/c10/core/TensorImpl: 1442

```
/**
 * Whether or not the tensor should be negated
 */
inline bool is_neg() const {
  constexpr auto negative_ks = DispatchKeySet(DispatchKey::Negative);
  return key_set_.has_all(negative_ks);
}
```

【NOTE】不需要实现，功能由pytorch内部完成

is_conj

判断一个Tensor的DispatchKeySet中的Conjugate位是否被设置。

示例

代码块

```
1  import torch
2
3  a = torch.tensor((3, 4))
4  # 此时Conjugate bit没有被设置
5  print(torch.is_conj(a))
6
7  """
8  >>> False
9  """
10
```

```
# 设置一下Conjugate bit
b = torch._C._set_conj(b, True)
print(torch.is_conj(a))

"""
>>> True
"""
```

源码

Source file: pytorch/aten/src/ATen/core/TensorBase.h: 380

```
inline bool is_conj() const {
  return impl_->is_conj();
}
```

Source file: pytorch/c10/core/TensorImpl: 1393

```
/**
 * Whether or not the imaginary part of the tensor should be negated
 */
inline bool is_conj() const {
  constexpr auto conjugate_ks = DispatchKeySet(DispatchKey::Conjugate);
  return key_set_.has_all(conjugate_ks);
}
```

【NOTE】不需要实现，功能由pytorch内部完成

contiguous

参考信息

pytorch文档：https://docs.pytorch.org/docs/stable/generated/torch.Tensor.contiguous.html

示例

代码块

```
import torch

# 直接创建的Tensor默认都是连续的
a = torch.randn((3, 4))
```

```python
print(a.is_contiguous())

"""
>>> True
"""

# 转置操作只会交换size以及重新计算strides，此时内存不连续
# 调用contiguous方法重新生成新的Tensor，将原来的内存拷贝为连续内存
b = a.T
print(b.is_contiguous())
c = b.contiguous()
print(c.is_contiguous())

"""
>>> False
>>> True
"""
```

源码

Source file: pytorch/aten/src/ATen/native/TensorProperties.cpp: 115

```
Tensor contiguous(const Tensor& self, MemoryFormat memory_format) {
  if (self.is_contiguous(memory_format)) {
    return self;
  }
  TORCH_CHECK(
      memory_format != MemoryFormat::Preserve,
      "preserve memory format is unsupported by the contiguous operator");

  return self.clone(memory_format);
}
```

Source file: pytorch/aten/src/ATen/native/TensorProperties.cpp: 2131

通过contiguous方法调用clone方法传入的memory_format不会是Preserve，因此直接看else逻辑，这里首先创建空Tensor，并调用copy_方法拷贝src Tensor。

```cpp
Tensor clone(
    const Tensor& src,
    std::optional<c10::MemoryFormat> optional_memory_format) {
  auto memory_format = optional_memory_format.value_or(MemoryFormat::Preserve);
  Tensor self;
  if (memory_format == MemoryFormat::Preserve) {
    if (src.is_non_overlapping_and_dense()) {
      // Copy all strides, this is marginally faster than calling empty_like
      self = at::empty_strided_symint(
          src.sym_sizes(), src.sym_strides(), src.options());
    } else {
      self = at::empty_like(src);
    }
  } else {
    self = at::empty_like(src, src.options(), memory_format);
  }

  if (src._is_zerotensor()) {
    self.zero_();
  } else {
    self.copy_(src);
  }
  return self;
}
```

【NOTE】依赖copy_方法实现。

INT64模拟方案

其他厂商处理方法

1. 寒武纪，直接截断，转为int32或者float类型

9.2. Cambricon PyTorch训练

Q：使用Cambricon PyTorch训练时，出现类似warning：

```
torch/nn/modules/batchnorm.py:113: UserWarning: cast int64(double) to int(float) implicitly due to known MLU restrictions.
```

A：算子暂不支持64位。默认情况下，long或int64类型数据会截断为int32，double类型数据会截断为float类型。

用户可设置环境变量来禁止默认的截断：`export CATCH_ALLOW_64BIT_TO_32BIT_CASTING=0`。如果网络中出现64位数据时，计算会报错，此时，可使用Python函数将64位数据强制转换为32位。

2. 燧原 torch_gcu2.5

7.6. 已知问题

1. 底层软硬件无法支持64位类型，如I64，F64等，torch_gcu会做隐式转换，转为使用32位数据类型。可能会导致精度问题或数值溢出问题。在使用时会输出warning提示信息。
2. 由于1，当tensor为64位类型时，使用tensor.data_ptr接口获取的地址可能有误

3. 摩尔线程GPU自身支持64位数据

4. 昇腾在torch_npu源码和架构介绍中可以知道 【昇腾学院】解密昇腾AI处理器--DaVinci架构（计算单元）-云社区-华为云 AI core不支持64位数据类型，会把一些涉及 64位计算op放在 AICPU(标量计算单元)上op_plugin/ops/base_ops/aclops/ArgsortKernelNpu.cpp · Ascend/op-plugin - Gitee.com，但是也有一些没有明显标识的算子支持int64 op_plugin/ops/base_ops/aclops/NonzeroKernelNpu.cpp · Ascend/op-plugin - 码云 - 开源中国 看不到算子内部实现，无法确认实现方案。
5. NV https://forums.developer.nvidia.com/t/question-about-64-bit-integer-performance/64147 在这个回答中可以看到，在Ampere架构之前都不原生支持64位宽的数据计算，而是通过软件模拟，拆分为高32和低32位，利用已有的 INT32 单元分步计算。Ampere之后原生支持64位计算。

SIPU可参考方案

方案 1：使用 int32 模拟 int64 （类似NV Ampere 架构之前，需要在kernel中处理）

用两个 int32 变量拼接表示一个 int64 数值，通过算法实现四则运算和比较，问了算子那边可以做，但是比较麻烦，可以利用SIMD特性，但是指令数量会变多，性能会有损耗

方案 2：限制数据范围（提前在框架中进行截断）

将 `int64` 张量替换为 `int32` （需确认数值范围不超过 2^31-1）

方案 3：使用RV Core计算

类似华为的方案，RV core目前支持64位的scalar计算，使用cpu的逻辑方式编写kernel，但是无法利用SIMD的特性，性能较差。

目前场景分析

int32 最大取值-2147483648 到 2147483647

1. bincount：统计每一个layer中被选中的专家编号，专家最大为256，可以控制在int32范围内
2. add_func

代码块

```
/share_data/users/haoliu2/miniconda3/envs/SIPU/lib/python3.10/site-packages/transformers/generation/utils.py
    if model_kwargs.get("use_cache", True):
        model_kwargs["cache_position"] = model_kwargs["cache_position"][-1:] + num_new_tokens
    else:
        past_positions = model_kwargs.pop("cache_position")
        new_positions = torch.arange(
```

```
            past_positions[-1] + 1, past_positions[-1] + num_new_tokens + 1, dtype=past_positions.dtype
        ).to(past_positions.device)
        model_kwargs["cache_position"] = torch.cat((past_positions, new_positions))
    return model_kwargs
```

model_kwargs["cache_position"] 和 num_new_tokens中保存的数据最大值取决于模型的最大上下文长度 config.max_position_embeddings 与模型支持的上下文长度相关，目前支持上下文长度最大的模型是Gemini ，达到百万级别，完全在int32的范围以内，不存在溢出风险

3. mul_func：

代码块

```
/share_data/users/haoliu2/miniconda3/envs/SIPU/lib/python3.10/site-packages/transformers/generation/utils.py
# finished sentences should have their next token be a padding token
if has_eos_stopping_criteria:
    next_tokens = next_tokens * unfinished_sequences + pad_token_id * (1 - unfinished_sequences)
```

- 对未完成的序列，保留模型预测的下一个令牌（`next_tokens`）。
- 对已完成的序列，用填充令牌（`pad_token_id`）替换预测结果。

理论最大值：`max(vocab_size - 1, pad_token_id)`

- GPT-2：词汇表大小 50,257 → 最大令牌ID = 50,256
- LLaMA-2：词汇表大小 32,000 → 最大令牌ID = 31,999
- BERT：词汇表大小 30,522 → 最大令牌ID = 30,521
- DS V3: 词汇表大小 129,280 → 最大令牌ID = 129,279

`pad_token_id` 通常较小（如 `0`），不会显著影响最大值

可以分析得出max取值在int32的范围以内，不存在溢出风险

目前推荐在框架侧进行处理，转换到int32来处理