Median函数

1. **函数功能**

   返回输入张量的中位数，如果输入张量的元素是偶数个，则将两个中间数较小的那个作为中位数返回。

2. **速看Pytorch中median源码**

```
// this does not reduce to median with dim because we don't want to copy twice
Tensor median_cpu(const Tensor& self) {
  NoNamesGuard guard;
  TORCH_CHECK(self.numel() > 0, "median cannot be called with empty tensor");
  if (self.dim() == 0 && self.numel() == 1) {
    return self.clone(at::MemoryFormat::Contiguous);
  }
  auto tmp_values = self.clone(at::MemoryFormat::Contiguous).view(-1);
  auto result = at::empty({1}, self.options());
  AT_DISPATCH_ALL_TYPES(self.scalar_type(), "median", [&] {
    // note, quick_select is 0 based while kthvalue is not
    int64_t k = (tmp_values.size(0) - 1) / 2;
    auto val_accessor = tmp_values.accessor<scalar_t, 1>();
    quick_select_template(
        val_accessor,
        k,
        [](scalar_t x, scalar_t y) -> bool {
          return ((_isnan<scalar_t>(x) && !_isnan<scalar_t>(y)) || (x > y));
        },
        [&](int64_t i, int64_t j) {
          std::swap(val_accessor[i], val_accessor[j]);
        });
    result.fill_(tmp_values[k]);
  });
  return result.view({});
}
```

3. **median源码具体分析**

   - 首先检查传入的张量self是否合法

     ```
     TORCH_CHECK(self.numel() > 0, "median cannot be called with empty tensor");
     ```

   - 接着 直接返回 或者 复制到临时数组tmp_values用以计算中位数

     ```
     if (self.dim() == 0 && self.numel() == 1) {
       return self.clone(at::MemoryFormat::Contiguous);
     }
     auto tmp_values = self.clone(at::MemoryFormat::Contiguous).view(-1);
     ```

   - 其次是 AT_DISPATCH_ALL_TYPES在Pytorch中的具体实现如下，大致为实现每一种数据类型的运算

```
#define AT_DISPATCH_ALL_TYPES(TYPE, NAME, ...)                                  \
  [&] {                                                                         \
    const auto& the_type = TYPE;                                                \
    /* don't use TYPE again in case it is an expensive or side-effect op  */ \
    at::ScalarType _st = ::detail::scalar_type(the_type);                       \
    RECORD_KERNEL_FUNCTION_DTYPE(NAME, _st);                                    \
    switch (_st) {                                                                \
      AT_PRIVATE_CASE_TYPE(NAME, at::ScalarType::Byte, uint8_t, __VA_ARGS__)    \
      AT_PRIVATE_CASE_TYPE(NAME, at::ScalarType::Char, int8_t, __VA_ARGS__)     \
      AT_PRIVATE_CASE_TYPE(NAME, at::ScalarType::Double, double, __VA_ARGS__)   \
      AT_PRIVATE_CASE_TYPE(NAME, at::ScalarType::Float, float, __VA_ARGS__)     \
      AT_PRIVATE_CASE_TYPE(NAME, at::ScalarType::Int, int32_t, __VA_ARGS__)     \
      AT_PRIVATE_CASE_TYPE(NAME, at::ScalarType::Long, int64_t, __VA_ARGS__)    \
      AT_PRIVATE_CASE_TYPE(NAME, at::ScalarType::Short, int16_t, __VA_ARGS__)   \
      default:                                                                  \
        AT_ERROR(#NAME, " not implemented for '", toString(_st), "'");          \
    }                                                                           \
  }()
```

- 最后是功能实现。采用lambda表达式编写median的功能并作为函数参数传入AT_DISPATCH_ALL_TYPES。在功能实现中调用quick_select_template，具体内核为快速排序，k用于选择第几个，median中的k则设置为中位数，即(size - 1) / 2。最后将temp_values[k]传入result，用于输出。

```
AT_DISPATCH_ALL_TYPES(self.scalar_type(), "median", [&] {
  // note, quick_select is 0 based while kthvalue is not
  int64_t k = (tmp_values.size(0) - 1) / 2;
  auto val_accessor = tmp_values.accessor<scalar_t, 1>();
  quick_select_template(
      val_accessor,
      k,
      [](scalar_t x, scalar_t y) -> bool {
        return ((_isnan<scalar_t>(x) && !_isnan<scalar_t>(y)) || (x > y));
      },
      [&](int64_t i, int64_t j) {
        std::swap(val_accessor[i], val_accessor[j]);
      });
  result.fill_(tmp_values[k]);
});
```

4. **算子实现接口分析**

def median(x, y, indices, global_median, axis=0, keepdim=False, kernel_name=“median”)

参数	类型	说明
x	dict	输入的张量
y	dict	输出的张量，是输入张量在某一维度的每行的中位数的值。
indices	dict	输出的张量，是输入张量在某一维度的每行的中位数在给定维度上的索引。 如果global_median为True，没有这个输出。
global_median	bool	属性，是否是计算整个tensor的中位数
axis	int	给定的维度，不能超过输入张量的维度。
keepdim	bool	输出张量是否与输入张量的维度数量保持一致，默认值为False。
kernel_name	str	kernel的名称

- indices用于记录中位数的索引
- global_median和axis用于划分范围，全局或者是某一维度

5. **具体实现方案**

该算子功能为返回输入张量的中位数，可以使用 Vector 单元进行加速。计算过程如下：

1. 使用 vmrgsort4 指令对输入数据进行排序
2. 排序结束后使用 vgather 指令选取中位数进行输出

该算子的 global_median 为 True， 属性 dim/keepdim 都不输入

- 由于对vmrgsort4和vgather指令并不了解，大概知道它是用于排序和挑选中位数合并。