basic Parallel computing

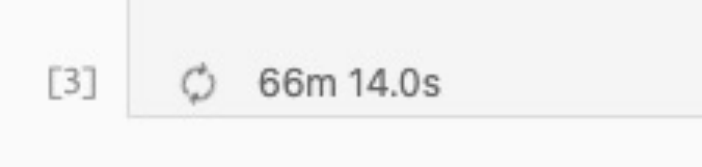
— PA1 as an example

Qihang

```python
    for b in range(B):  # Each sample in batch
        for d in range(D):  # Each filter
            for i in range(H_out):  # Output height
                for j in range(W_out):  # Output width
                    h_start = i * stride
                    w_start = j * stride
                    for c in range(C_img):  # Each input channel
                        for ki in range(K):  # Kernel row
                            for kj in range(K):  # Kernel col
                                h = h_start + ki - padding
                                w = w_start + kj - padding
                                if 0 <= h < H and 0 <= w < W:  # Ensure index is within bounds
                                    grad_img[b, c, h, w] += grad_out[b, d, i, j] * filter[d, c, ki, kj]

    return grad_img, grad_filter
```

7 for loops



For loops on GPU can even be slower than CPU

RTX 4090 : 82.6 TFLOPS (float32) 2.23 GHz 512 tensor cores
Intel i7-14700k: 1.2 TFLOPS (float32) 3.4 GHz 20 cores

1. broadcast

```
import torch
a = torch.tensor([[1,2], [3,4]])
b = torch.tensor([[1,-1]])
c = torch.tensor([[1,-1], [1,-1]])
print('a_info:\n', a.shape,'\n', a, '\n'+"-"*20)
print('b_info:\n', b.shape,'\n', b, '\n'+"-"*20)
print('c_info:\n', c.shape,'\n', c, '\n'+"-"*20)
ab = a + b
ac = a + c
print('ab:\n', ab)
print('ac:\n', ac)
print('if ab == ac', ab == ac)
```

```
a_info:
 torch.Size([2, 2])
 tensor([[1, 2],
        [3, 4]])
--------------------
b_info:
 torch.Size([1, 2])
 tensor([[ 1, -1]])
--------------------
c_info:
 torch.Size([2, 2])
 tensor([[ 1, -1],
        [ 1, -1]])
--------------------
ab:
 tensor([[2, 1],
        [4, 3]])
ac:
 tensor([[2, 1],
        [4, 3]])
if ab == ac tensor([[True, True],
        [True, True]])
```

2. rearrange

```
# rearrange
import torch
from einops import rearrange

a = torch.tensor([[1,2], [3,4]])
b = rearrange(a, 'h w -> w h')
c = rearrange(a, 'h w -> 1 1 h w')

print('a_info:\n', a.shape,'\n', a, '\n'+"-"*20)
print('b_info:\n', b.shape,'\n', b, '\n'+"-"*20)
print('c_info:\n', c.shape,'\n', c, '\n'+"-"*20)
```

```
a_info:
 torch.Size([2, 2])
 tensor([[1, 2],
        [3, 4]])
--------------------
b_info:
 torch.Size([2, 2])
 tensor([[1, 3],
        [2, 4]])
--------------------
c_info:
 torch.Size([1, 1, 2, 2])
 tensor([[[[1, 2],
          [3, 4]]]])
--------------------
```

3. device

```
# device
import torch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```



```
# the defalut device will be cpu
a = torch.tensor([[1.,2.], [3.,4.]])
print('a.device', a.device)

a = a.to(device)
print('a.device', a.device)
```

```
a.device cpu
a.device cuda:0
--------------------
```

```
# if the tensor and the model are not on the same device,
# the output will be on the the same device
model = torch.nn.Linear(2, 1)
model.to(device)
print('-'*20)
print('model.device', next(model.parameters()).device)
print('a.device', a.device)
output_a = model(a)
print('output_a.device', output_a.device)
```

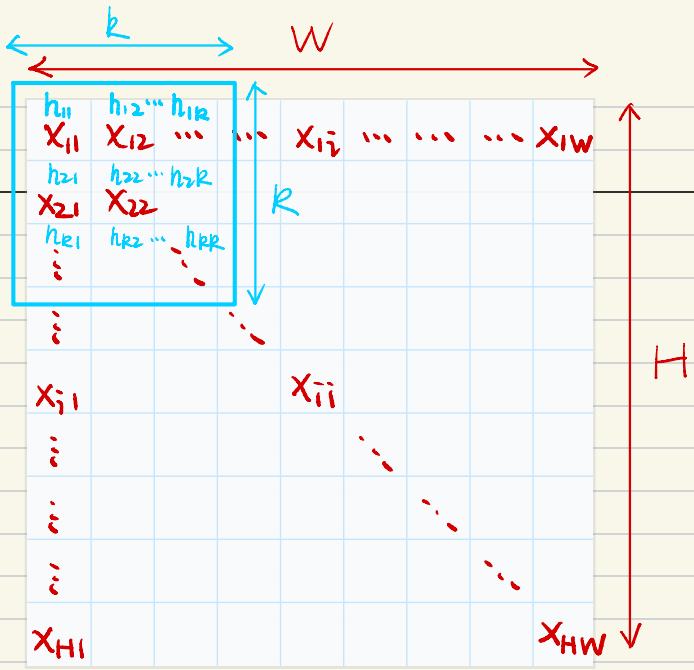
```
model.device cuda:0
a.device cuda:0
output_a.device cuda:0
--------------------
```

```
#else if the tensor and the model are not on the same device,
# their will be an error
b = torch.tensor([[1.,2.], [3.,4.]])
print('-'*20)
print('model.device', next(model.parameters()).device)
print('b.device', b.device)
output_b = model(b)
print('output_b.device', output_b.device)
```

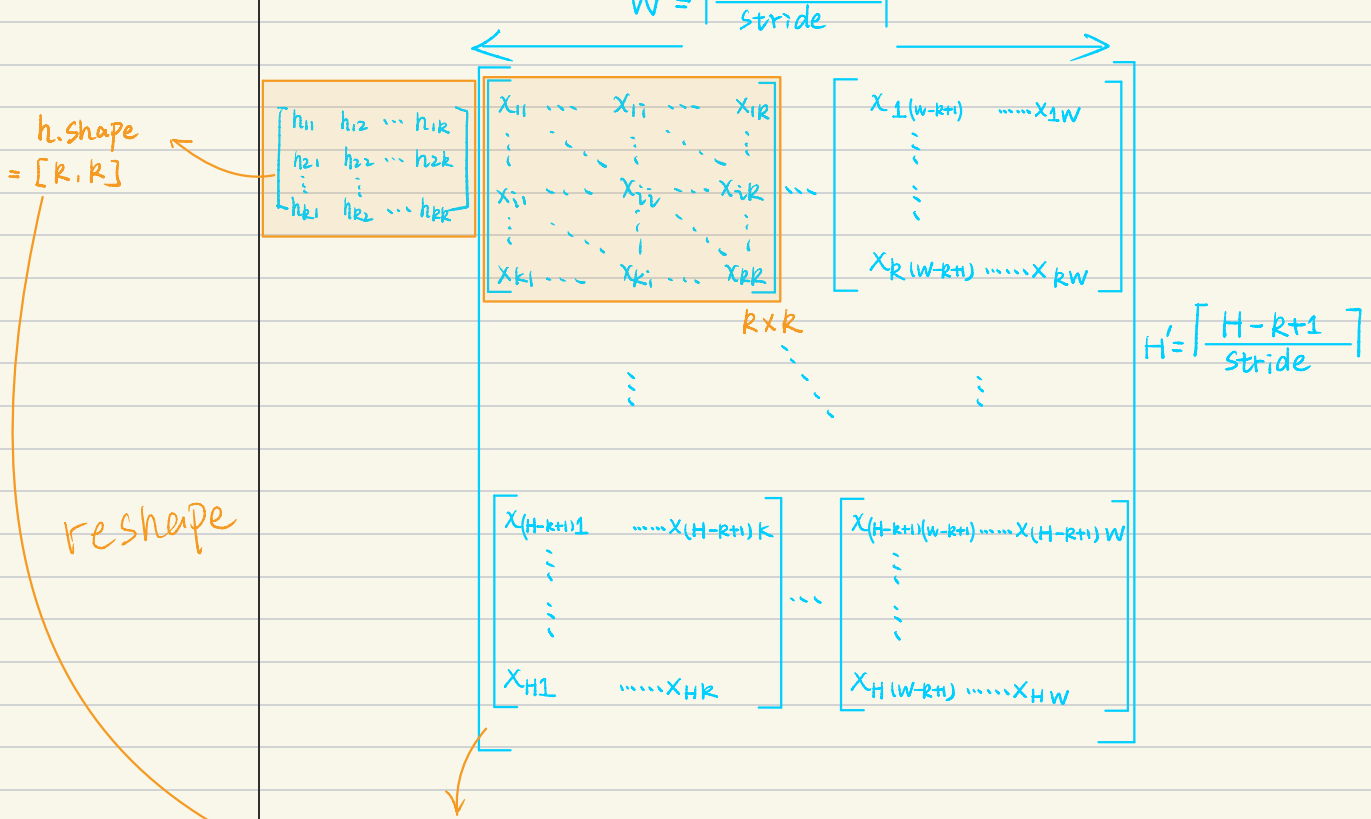
```
model.device cuda:0
b.device cpu
```

RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cuda:0 and cpu! (when checking argument for argument mat1 in method wrapper_CUDA_addmm)

Image to column



$$W' = \left\lceil \frac{W-k+1}{stride} \right\rceil$$



reshape

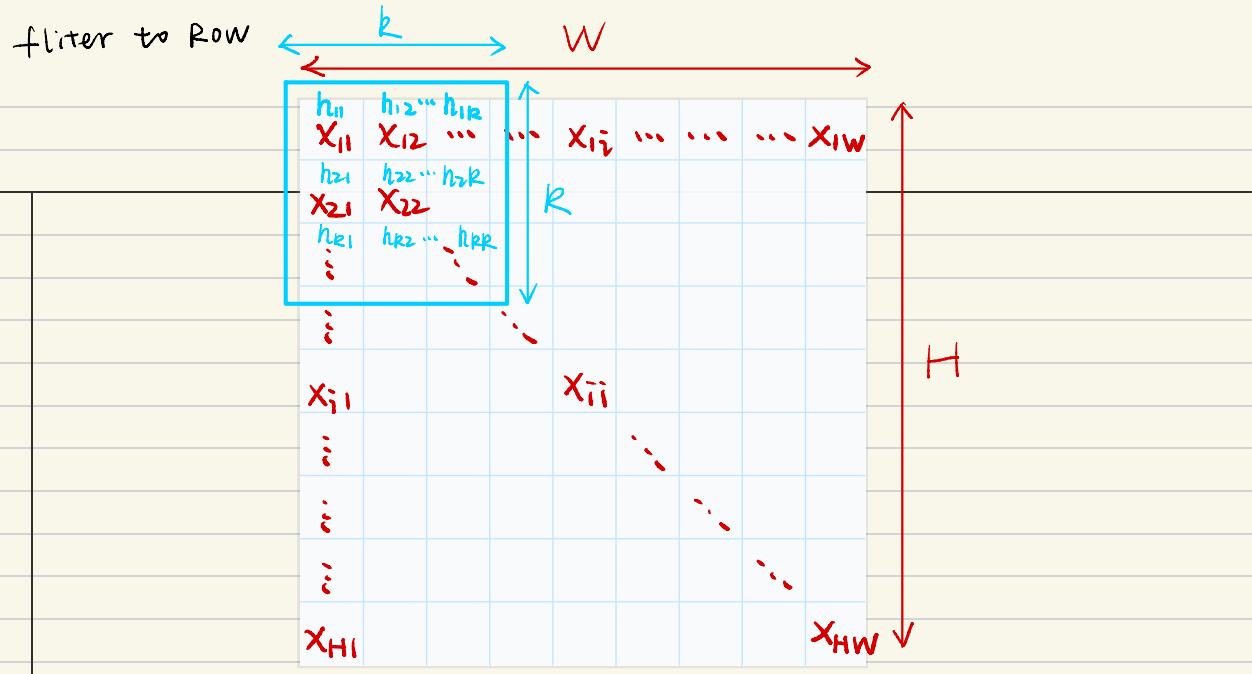
$X\_col.shape = [H', W', k, k] \longrightarrow [H' \times W', k \times k]$

$h.shape = [1, 1, k, k] \longrightarrow [1, k \times k]$

broadcast $(h * x\_col).shape = [H', W', k, k]$

sum over last 2 dim

$$y^\top = (h * x)^\top = \begin{bmatrix} h_m & h_{m-1} & \cdots & h_3 & h_2 & h_1 \end{bmatrix} \begin{bmatrix} x_{m-\lfloor m/2 \rfloor} & x_{m-\lfloor m/2 \rfloor+1} & \cdots & x_m & x_{m+1} & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & x_{m-1} & x_m & \cdots & \vdots & \vdots \\ x_1 & x_2 & \cdots & \vdots & x_{m-1} & \cdots & x_n & 0 \\ 0 & x_1 & \cdots & \vdots & \vdots & \cdots & x_{n-1} & x_n \\ \vdots & 0 & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & x_1 & x_2 & \cdots & x_{n-\lfloor m/2 \rfloor+1} & x_{n-\lfloor m/2 \rfloor} \end{bmatrix}.$$



$[H, W]$

$W' = \left\lceil \frac{W-k+1}{stride} \right\rceil$

$H' = \left\lceil \frac{H-k+1}{stride} \right\rceil$

conv

conv

conv

conv

$X.shape = [H, W]$

Broad Cast

$X * h\_row$

$h\_row.shape = [H', W', H, W]$

$X.shape = [1, 1, H, W]$

$[H' \times W', H \times W]$

$[1, H \times W]$

$$y = h * x = \begin{bmatrix} h_{\lfloor m/2 \rfloor+1} & h_{\lfloor m/2 \rfloor+2} & \cdots & h_m & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ h_{\lfloor m/2 \rfloor} & h_{\lfloor m/2 \rfloor+1} & \cdots & h_{m-1} & h_m & 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_1 & h_2 & \cdots & \cdots & \cdots & \cdots & h_m & 0 & \cdots & 0 \\ 0 & h_1 & h_2 & \cdots & \cdots & \cdots & \cdots & h_m & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & \cdots & 0 & h_1 & h_2 & \cdots & h_{\lfloor m/2 \rfloor+2} \\ 0 & 0 & \cdots & \cdots & & \cdots\cdots & 0 & h_1 & \cdots & h_{\lfloor m/2 \rfloor+1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

b = ~~~

c = copy(b)

a = deepcopy(b)