Direct convolution

Original code

```
def direct_conv3d(direct_input, direct_kernel, direct_output):
    for n in range(0, batchsize): # Preferrably go to different blocks, less data sharing

        for d in range(0, outdepth):
            for h in range(0, outheight):
                for w in range(0, outwidth):

                    for oc in range(0, outchannels): # Preferrably go to different blocks,
less data sharing

                        # Compulsory serial code starts
                        for ic in range(0, inchannels): # all should preferrably go to the
same block, but too much for a single thread

                            # Don't divide this between threads
                            for kd in range(0, kdim):
                                for kh in range(0, kdim):
                                    for kw in range(0, kdim):
                                            direct_output[n, d, h, w, oc] = direct_output
[n, d, h, w, oc] + direct_input[ n, d+kd, h+kh, w+kw, ic]*direct_kernel[oc, kd, kh, kw, i
c]
```

Merging dhw

```
def direct_conv3d(direct_input, direct_kernel, direct_output):
    DHW = outdepth*outheight*outwidth
    HW = outheight*outwidth
    for n in range(0, batchsize): # Must go to different blocks, less data sharing
        for dhw in range(0, DHW):
            for oc in range(0, outchannels): # Must go to different blocks, less data shar
ing
                d, dhw_residual = divmod(dhw, HW)
                h, w = divmod(dhw_residual, outheight)
                # Compulsory serial code starts
                for ic in range(0, inchannels): # all must go to the same block, but too m
uch for a single thread
                    # Don't divide this between threads. Per thread it can compute atleast
this
                    for kd in range(0, kdim):
                        for kh in range(0, kdim):
```

```
                            for kw in range(0, kdim):
                                    direct_output[n, d, h, w, oc] = direct_output[n, d, h,
w, oc] + direct_input[ n, d+kd, h+kh, w+kw, ic]*direct_kernel[oc, kd, kh, kw, ic]
```

Block tiling

```
# Tiling direct convolution
def direct_conv3d(direct_input, direct_kernel, direct_output):
    DHW = outdepth*outheight*outwidth
    HW = outheight*outwidth
    for cta_n in range(0, batchsize, CTAtile_N):
        for cta_dhw in range(0, DHW, CTAtile_DHW):
            for cta_oc in range(0, outchannels, CTAtile_OC):
                # Work allocated for a block
                for n in range(0, CTAtile_N):
                    for dhw in range(0, CTAtile_DHW):
                        for oc in range(0, CTAtile_OC):
                            d, dhw_residual = divmod(dhw+cta_dhw, HW)
                            h, w = divmod(dhw_residual, outheight)
                            # Compulsory serial code starts
                            for ic in range(0, inchannels): # all must go to the same bloc
k, but too much work for a single thread
                                # Don't divide this between threads. Per thread it can com
pute atleast this
                                for kd in range(0, kdim):
                                    for kh in range(0, kdim):
                                        for kw in range(0, kdim):
                                                direct_output[n+cta_n, d, h, w, oc+cta_oc]
= direct_output[n+cta_n, d, h, w, oc+cta_oc] + direct_input[ n+cta_n, d+kd, h+kh, w+kw, i
c]*direct_kernel[oc+cta_oc, kd, kh, kw, ic]
```

Now per block we have work of: `CTAtile_N X CTAtile_DHW X CTAtile_OC X IC` .

The innermost 3 loops with kdim are per thread. So that doesn't need to be tiled.

This has to be divided into per warps work.

Merge the dimension `CTAtile_DHW X IC` together, because of **NDHWC** layout.

n, oc are tiled trivially like the implicit gemm implementation.

Tiling 2 levels

```
CTAtile_N = 4
CTAtile_DHW = 4 # This should divide outdepth, outheight, outwidth individually otherwise
 the indexing gets tricky.
```

```
CTAtile_OC = 8

WARPtile_N = 2
WARPtile_OC = 4
WARPtile_DHWIC = 8
DHW = outdepth*outheight*outwidth
HW = outheight*outwidth
CTAtile_DHW_IC = CTAtile_DHW*inchannels

def direct_conv3d(direct_input, direct_kernel, direct_output):

    for cta_n in range(0, batchsize, CTAtile_N):
        for cta_dhw in range(0, DHW, CTAtile_DHW):
            for cta_oc in range(0, outchannels, CTAtile_OC):

                # Work allocated for a block
                for warp_n in range(0, CTAtile_N, WARPtile_N):
                    for warp_oc in range(0, CTAtile_OC, WARPtile_OC):
                        for warp_dhwic in range(0, CTAtile_DHW_IC, WARPtile_DHWIC):

                            for n in range(0, WARPtile_N):
                                for oc in range(0, WARPtile_OC):
                                        for dhw_ic in range(0, WARPtile_DHWIC): # all must
go to the same block, but too much work for a single thread

                                            dhw, ic = divmod(dhw_ic+warp_dhwic, inchannel
s)
                                            d, dhw_residual = divmod(dhw+cta_dhw, HW)
                                            h, w = divmod(dhw_residual, outheight)
                                            for kd in range(0, kdim):
                                                for kh in range(0, kdim):
                                                    for kw in range(0, kdim):
                                                            direct_output[n+cta_n+warp_n,
 d, h, w, oc+cta_oc+warp_oc] = direct_output[n+cta_n+warp_n, d, h, w, oc+cta_oc+warp_oc] +
direct_input[ n+cta_n+warp_n, d+kd, h+kh, w+kw, ic]*direct_kernel[oc+cta_oc+warp_oc, kd, k
h, kw, ic]
```