Chase Jones
Dr. Muyan
CSC 196u
1 Nov 2018

Convolution questions

1) Name 3 applications of convolution.

Noise reduction in audio processing. Blurring images. Amplifying a certain color in video processing.

2) How many floating operations are being performed in your convolution kernel in terms of variables that indicate mask width, input image size, and number of channels (utilize the same variable names used in the template.cu e.g., utilize MASK_WIDTH to indicate mask width)? Explain.

We do `(MASK_WIDTH*MASK_WIDTH)*imageChannels` fp multiplications and `(MASK_WIDTH*MASK_WIDTH)*imageChannels` fp additions per output element (for each value computed plus one more for adding the first element to the variable `output` which = 0 initially). We have `imageChannels*imageWidth*imageHeight` elements overall, so the final equation for number of fp operations is:

`((MASK_WIDTH*MASK_WIDTH)` *[multiplications]* `+ (MASK_WIDTH*MASK_WIDTH)` *[additions]*`) * imageChannels*imageWidth*imageHeight`

3) How many global memory reads are being performed by your kernel in terms of variables that indicate output tile width, mask width, input image size, and number of channels (utilize the same variable names used in the template.cu e.g., utilize O_TILE_WIDTH to indicate output tile width)? explain. Assume that ghost elements also require global reads.

For each thread, we load 3 values, or imageChannels per pixel, and each thread computes one pixel. We load each element at least once when it is in the block coordinates, since each block

loads all the values it needs, we can guarantee that at most, the border elements of tiles will be loaded 3 more times. Thus we load

```
imageChannels*imageHeight*imageWidth
 [initial loads] +
(((MASK_WIDTH-1)/2)^2 *
(ceil(imageWidth/O_TILE_SIZE) + 1) *
(ceil(imageHeight/O_TILE_SIZE) + 1))
[corner elements that are repeated] +
(((MASK_WIDTH-1)/2)* O_TILE_WIDTH * 4)
[side elements that are repeated]
```

4) How many global memory writes are being performed by your kernel in terms of variables that indicate input image size and number of channels (utilize the same variable names used in the template.cu e.g., utilize imageWidth to indicate image width)? Explain.

Since we write each element only once (where an element is one channel of one pixel) , the number of global writes is equal to:
`imageChannels*imageWidth*imageHeight`

5) What do you think happens as you increase the mask width (e.g., you increase it from 5 to 1024) in your kernel while you keep your output tile width at 16? What do you end up spending most of your time doing? Do you think using a tiled convolution approach would still be a good solution in this case? Hint: think of the shared memory size.

In this case, where shared memory is equal to `[BLOCK_WIDTH][BLOCK_WIDTH]`, where `BLOCK_WIDTH = O_TILE_WIDTH+MASK_WIDTH-1`, we see that shared memory would scale with the mask width, so the non-increase in output tile does not effect the shared memory in a negative way.

However, the shared memory hits its cap. The width of the memory becomes 16+1024-1 = 1039. This means we have 1039^2 = 1079521 elements are loaded into memory, with size of 4 bytes per elements. We'd need a shared memory of size 4318084 bytes, or a 4.32 MB shared memory. Sadly, our memory is only 48KB, so we cannot load all the elements needed into our shared memory, meaning we'd spend most of our time loading elements from main memory.

Tiled convolution would still be a better choice than not using any shared memory, but this approach with a large mask size like 1024 would be very inefficient in terms of memory accesses per thread.

6) Do you have to have a separate output memory buffer, e.g., called P, in your kernel? Put it in another way, why can't you perform the convolution in place (by just using the input memory buffer, e.g. called N)?

Yes, another buffer is needed for outputs. If only one buffer were used (meaning output would overwrite the input), we would be overwriting the unmodified values needed to calculate other values. For instance, if we have a 1d input {1,2,3} and a mask of 3 {1,2,1}, as we go through, we'd overwrite each of the input values, {4, 2, 3} is now the input/output after the first value is calculated, which means {4, 11, 3} would be the value of the input/output after 2, when it should have been the original input for the input and {4, 8, 3 } for the output.

7) What is the identity mask? What would be the values in it for a mask width of 5 for 2D convolution?

The identity mask is a mask which returns only the center input value. For a mask of 5, the mask would be {0, 0, 1, 0, 0}