Parallel computing
Lab 4: Convolution

Name	ID	SEC	BN
Norhan Reda Abdelwahed Ahmed	9203639	2	31
Hoda Gamal Hamouda Ismail	9203673	2	33

Approach:

Same tile size = 16 for k2 & k3
K1: dim3 threadsPerBlock(16,16,1);
K2: dim3 threadsPerBlock(16+(maskdim - 1),16+(maskdim - 1),1);
K3: dim3 threadsPerBlock(16,16,1);
All Ks: dim3 numBlocks((width + tile - 1) / tile , (height + tile - 1) / tile , batch_size);

Batch size = 4 , Image size = 1024 * 1024 * 3

Mask in constant memory using `restrict`

Mask dim	k1	k2	k3	Pytorch GPU	Pytorch CPU (s)
3	1.5208ms	912.34us	1.3206ms	1.053ms	0.20866966247558594
5	2.8107ms	1.9225ms	2.3094ms	2.605ms	0.23140478134155273
7	5.2453ms	3.6349ms	4.1718ms	4.611ms	0.37041521072387695
9	8.6661ms	7.1195ms	6.7604ms	6.909ms	0.8478844165802002
11	12.817ms	11.124ms	9.9182ms	10.072ms	0.8033547401428223
13	17.823ms	15.640ms	13.790ms	13.522ms	1.182936429977417
15	23.689ms	21.696ms	18.184ms	17.852ms	1.5017869472503662

Mask <u>not</u> in constant memory using `restrict`
Values of cuda kernels times <u>almost didn't change</u>

Mask in constant memory using `cudaMemcpyToSymbol`

Mask dim	k1	k2	k3	Pytorch GPU Same as above	Pytorch CPU (s) Same as above
3	1.4943ms	821.13us	1.2532ms	1.053ms	0.20866966247558594
5	2.4365ms	1.5649ms	2.0486ms	2.605ms	0.23140478134155273
7	4.4147ms	2.7181ms	3.6295ms	4.611ms	0.37041521072387695
9	6.1811ms	5.5570ms	5.8611ms	6.909ms	0.8478844165802002
11	9.1638ms	8.3201ms	8.5704ms	10.072ms	0.8033547401428223
13	12.081ms	11.478ms	11.904ms	13.522ms	1.182936429977417
15	16.208ms	15.564ms	15.674ms	17.852ms	1.5017869472503662

Comment:

All kernels:

- K1 is the worst one in all cases
- K2 is better than k3 in small mask sizes, then in large mask sizes k3 is better than k2 using `restrict`
- K2 is better than k3 or almost the same using `cudaMemcpyToSymbol`

K1 is the worst one:

It's because of accessing global memory many times
As for calculating one output pixel, each thread accesses the global memory for (maskdim*maskdim*3) times to get needed input pixels.

While in k2 & k3, we use shared memory and tiling approaches to make threads in the block contribute to get all needed input pixels from global memory
((tile size + maskdim -1)*(tile size + maskdim -1)*3),
then each thread calculates its output pixel using needed pixels (maskdim*maskdim*3).

Constant vs non-constant mask:

- Almost no difference when using `const float* __restrict__ mask` or not.
- Significant decrease in time when using `cudaMemcpyToSymbol`.

Our results vs pytorch:

- Time of pytorch in GPU is very close to our best of cuda 3 kernels when using `const * __restrict__`
  Sometimes our best kernel is faster and sometimes pytorch in GPU is faster
  With small differences, these difference increases with batch size and mask size

- Time of pytorch in GPU is still close to our best of cuda 3 kernels when using `cudaMemcpyToSymbol`
  But our best kernel is faster
  With a difference that increases with batch size and mask size

- But pytorch in CPU is much slower than all cuda 3 kernels with a significant difference.

Batch size = 50 , Image size = 1024 * 1024 * 3

Mask in constant memory using `restrict`

Mask dim	k1	k2	k3	Pytorch GPU	Pytorch CPU (s)
3	18.950ms	11.374ms	16.411ms	12.980ms	1.9651260375976562
5	35.023ms	23.937ms	28.714ms	32.295ms	4.004384517669678
7	65.403ms	45.299ms	51.991ms	57.213ms	5.26420259475708
9	108.13ms	88.863ms	84.307ms	85.752ms	7.806816101074219
11	147.07ms	130.92ms	114.07ms	125.053ms	12.150193214416504
13	222.36ms	195.14ms	172.11ms	168.351ms	15.831034183502197
15	225.32ms	217.05ms	193.80ms	199.077ms	20.415729761123657

Mask not in constant memory using `restrict`
Values of cuda kernels times almost didn't change

Mask in constant memory using `cudaMemcpyToSymbol`

Mask dim	k1	k2	k3	Pytorch GPU Same as above	Pytorch CPU (s) Same as above
3	18.617ms	10.185ms	15.575ms	12.980ms	1.9651260375976562
5	30.370ms	19.467ms	25.464ms	32.295ms	4.004384517669678
7	55.031ms	33.882ms	45.220ms	57.213ms	5.26420259475708
9	77.040ms	69.342ms	73.069ms	85.752ms	7.806816101074219
11	114.26ms	91.981ms	97.996ms	125.053ms	12.150193214416504
13	133.90ms	121.55ms	124.35ms	168.351ms	15.831034183502197
15	202.15ms	156.52ms	158.19ms	199.077ms	20.415729761123657

Comment:

With increasing batch size:

- We can obtain exactly the same conclusions as above.
- Time increases overall.

Batch size = 4 , Image size = 255* 255* 3

Mask in constant memory using `cudaMemcpyToSymbol`

Mask dim	k1	k2	k3	Pytorch GPU Same as above	Pytorch CPU (s) Same as above
3	78.560us	51.744us	81.311us	137.201ms	0.008887052536010742
5	145.34us	102.43us	136.06us	182.000us	0.014289379119873047
7	273.05us	176.03us	235.97us	314.000us	0.0355684757232666
9	394.05us	352.13us	378.43us	472.000us	0.0347590446472168
11	593.25us	526.08us	549.50us	689.000us	0.04861927032470703
13	783.29us	725.53us	758.84us	936.000us	0.07089519500732422
15	1.0560ms	986.68us	996.38us	1.233ms	0.09325647354125977

Batch size = 4 , Image size = 3000* 4000* 3

Mask in constant memory using `cudaMemcpyToSymbol`

Mask dim	k1	k2	k3	Pytorch GPU Same as above	Pytorch CPU (s) Same as above
3	17.457ms	9.2855ms	14.350ms	12.535ms	1.5423247814178467
5	28.299ms	17.815ms	23.269ms	30.514ms	2.9205591678619385
7	51.099ms	30.994ms	41.267ms	52.572ms	5.0790863037109375
9	70.236ms	63.359ms	65.081ms	78.835ms	7.100457668304443
11	107.89ms	95.604ms	98.246ms	115.144ms	9.925102710723877
13	127.02ms	131.74ms	136.46ms	153.115ms	13.183547258377075
15	186.83ms	168.51ms	174.89ms	181.916ms	17.17416214942932

Comment:

With changing image size:

- We can obtain exactly the same conclusions as above.
- Time increases overall when increasing image size and vice versa.

Approach:

Same tile size = 8 for k2 & k3

K1: dim3 threadsPerBlock(8,8,1);

K2: dim3 threadsPerBlock(8+(maskdim - 1),8+(maskdim - 1),1);

K3: dim3 threadsPerBlock(8,8,1);

All Ks: dim3 numBlocks((width + tile - 1) / tile , (height + tile - 1) / tile , batch_size);

Batch size = 4 , Image size = 1024 * 1024 * 3

Mask in constant memory using `cudaMemcpyToSymbol`

Mask dim	k1	k2	k3	Pytorch GPU Same as above	Pytorch CPU (s) Same as above
3	1.5769ms	961.08us	1.2118ms		
5	3.6161ms	1.8603ms	2.3000ms		
7	6.9138ms	4.1181ms	4.0622ms		
9	11.453ms	6.2351ms	6.0064ms		
11	16.893ms	11.355ms	9.0488ms		
13	23.344ms	15.536ms	12.360ms		
15	30.991ms	24.552ms	16.215ms		

Comment:

With decreasing block width:

- Time increases overall.
- K2 is better than k3 in small mask sizes, then in large mask sizes k3 is better than k2 using `cudaMemcpyToSymbol`

Approach:

Same block width = 32 for all Ks

K1: dim3 threadsPerBlock(32,32,1);

K2: dim3 threadsPerBlock(32,32,1); → tile = 32 - (maskdim - 1)

K3: dim3 threadsPerBlock(32,32,1); → tile = 32

All Ks: dim3 numBlocks((width + tile - 1) / tile , (height + tile - 1) / tile , batch_size);

Batch size = 4 , Image size = 1024 * 1024 * 3

Mask in constant memory using `cudaMemcpyToSymbol`

Mask dim	k1	k2	k3	Pytorch GPU Same as above	Pytorch CPU (s) Same as above
3	1.5356ms	761.37us	1.4818ms		
5	2.6230ms	1.4924ms	2.0477ms		
7	4.7348ms	2.6866ms	2.8684ms		
9	6.3275ms	4.3651ms	3.9755ms		
11	9.3334ms	6.8711ms	5.3436ms		
13	11.917ms	10.337ms	6.9984ms		
15	16.085ms	14.775ms	9.1582ms		

Comment:

With Increasing block width and same block width:

- Time decreases overall.
- K2 is better than k3 in small mask sizes, then in large mask sizes k3 is better than k2 using `cudaMemcpyToSymbol`
- Tile size within a block in k2 decreases with increasing mask sizes, while it's constant in k3, this decreases the output tile size calculated within a block.

Comments:

K2 is worse than k3 in large mask sizes
and this is more obvious in small block widths or large mask sizes:

It may be because of that there will be control divergence
As some threads within the warp after getting one input element will stop
and not contributing in calculating the output.
And this number of threads depends on the block width and mask size
(tile size = block width - (mask size -1)) where number of working threads = tile size