Parallel Computing

Name	Section	B.N.	ID
Mustafa Mahamoud Hamada	2	25	9203519
Karim Mahmoud Kamal	2	12	9203076

Mask stored in Const memory

All tests performed on mask size = **5 x 5**

Big Image (4252 * 2835)

	Kernal 1 (no tiling)	Kernal 2 (input tiling)	Kernal 3 (output tiling)
16 * 16	5.3202ms	4.3540ms	3.6121ms
32 * 32	4.2491ms	3.9658ms	3.9655ms
Python	0.055235862731933594 sec		

Small Image (1600 * 900)

	Kernal 1 (no tiling)	Kernal 2 (input tiling)	Kernal 3 (output tiling)
16 * 16	475.89us	445.72us	425.24us
32 * 32	524.92us	489.30us	489.88us
Python	0.01000071 sec		

Comments in case of Mask stored in Constant memory:

1- Always output tiling is the best kernel and followed by the input tiling and the no tiling is the worst.

2- In general, 32x32 is better than 16x16 as we increased the number of threads per block so execution time is decreased but in some cases the control divergence may cause the 16x16 is better than 32x32.

3- Always the python code is worst than our code and the reason explained in the general comments below.

Mask NOT stored in Const memory

All tests performed on mask size = **5 x 5**

Big Image (4252 * 2835)

	Kernal 1 (no tiling)	**Kernal 2 (input tiling)**	**Kernal 3 (output tiling)**
16 * 16	6.1682ms	5.6607ms	4.5496ms
32 * 32	4.5085ms	5.1774ms	5.0583ms
Python	0.055235862731933594 sec		

Small Image (1600 * 900)

	Kernal 1 (no tiling)	**Kernal 2 (input tiling)**	**Kernal 3 (output tiling)**
16 * 16	573.14us	672.37us	539.99us
32 * 32	556.76us	635.67us	620.05us
Python	0.01000071 sec		

Comments in case of NO Const memory:

1- Always output tiling is the best kernel in case of 16x16.

2- In some cases, the no tiling is better than input tiling and output tiling because of control divergence which is explained below.

3- In general, 32x32 is better than 16x16 as we increased the number of threads per block except in case of output tiling because of the control divergence may cause the 16x16 is better than 32x32.

Mask stored in Const memory

All tests performed on mask size = **11 x 11**

Big Image (4252 * 2835)

	Kernal 1 (no tiling)	Kernal 2 (input tiling)	Kernal 3 (output tiling)
16 * 16	29.188ms	35.448ms	13.257ms
32 * 32	18.773ms	19.031ms	13.339ms
Python	0.07000470		

Small Image (1600 * 900)

	Kernal 1 (no tiling)	Kernal 2 (input tiling)	Kernal 3 (output tiling)
16 * 16	2.6512ms	4.1930ms	1.5714ms
32 * 32	2.2778ms	2.2798ms	1.6074ms
Python	0.00997305		

Comments in case of Increasing mask size to 11x11 while all tests while using constant memory:

1- In general, 32x32 is better than 16x16 as we increased the number of threads per block except in case of output tiling because of the control divergence may cause the 16x16 is better than 32x32

2- The output tiling is always the better across all kernels when mask size increased from 5x5 to 11x11.

3- In input tiling, not all threads contributing in the calculations so that no tiling is better than the input tiling (take less time in execution).

General Comments:

Image Size Impact: Larger images generally result in longer execution times across all kernels and configurations. This is expected, as processing larger amounts of data requires more computational resources and time.

Always Prefer 32x32 Tile Size Over 16x16, Except in Some Cases: With a 16x16 tile size, there are fewer threads within a warp compared to a 32x32 tile size. With fewer threads per warp, there is less potential for control divergence because all threads within a warp are more likely to follow the same execution path Therefore opting for a smaller tile size like 16x16 can help mitigate the effects of control divergence and improve parallel efficiency compared to a larger tile size like 32x32.

Impact of Kernel Fetching from Constant Memory: The execution time of kernels fetching mask values from constant memory in the GPU is generally less than for kernels fetching mask values from global memory. This is intuitive as constant memory is closer to SM units than global memory.

Kernel Size Influence: As the mask size increases from 5x5 to 11x11, the execution time also increases. This correlation is expected.

Python Execution Time Compared to C Code: In Python, the execution time is consistently worse than that of optimized C code. This is because libraries implemented in Python, compared to C, can be slower due to interpretation overhead. C code can be highly optimized for performance, especially for tasks involving heavy computation, as C compilers can generate efficient machine code. Additionally, in C, there's more control over memory management, contributing to performance advantages.