GPU programming - assignment 3
Dr. Cherif Salama

Team members:

1. Mina Ashraf Gamil - 900182973
2. Mohamed Ayman Mohamed - 900182267

GPU Model: GeForce RTX 3090

```
Fri Nov  4 19:33:12 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 520.61.05    Driver Version: 520.61.05    CUDA Version: 11.8     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  On   | 00000000:02:00.0  On |                  N/A |
|  0%   41C    P8    25W / 350W |    366MiB / 24576MiB |      6%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A       950      G   /usr/lib/xorg/Xorg                194MiB |
|    0   N/A  N/A      1459      G   /usr/bin/gnome-shell               38MiB |
|    0   N/A  N/A    224141      G   ...RendererForSitePerProcess      131MiB |
+-----------------------------------------------------------------------------+
```

```
cse-p07-g07f@csep07g07f:~/GPU_programming_with_CUDA$ nvidia-smi --query-gpu=name
 --format=csv,noheader
NVIDIA GeForce RTX 3090
```

GeForce and TITAN Products

GPU	Compute Capability
GeForce RTX 3090 Ti	8.6
GeForce RTX 3090	8.6

We are using the 8.6
For the full-table, check this link:
https://docs.nvidia.com/cuda/cuda-c-programming-guide/#features-and-technical-specifications

Note that the KB and K units used in the following table correspond to 1024 bytes (i.e., a KiB) and 1024 respectively.

Table 15. Technical Specifications per Compute Capability

	Compute Capability															
Technical Specifications	**3.5**	**3.7**	**5.0**	**5.2**	**5.3**	**6.0**	**6.1**	**6.2**	**7.0**	**7.2**	**7.5**	**8.0**	**8.6**	**8.7**	**8.9**	**9.0**
Maximum number of resident grids per device (Concurrent Kernel Execution)	32				16	128	32	16	128	16	128					
Maximum dimensionality of grid of thread blocks	3															
Maximum x-dimension of a grid of thread blocks	$2^{31}-1$															
Maximum y- or z-dimension of a grid of thread blocks	65535															
Maximum dimensionality of a thread block	3															
Maximum x- or y-dimension of a block	1024															
Maximum z-dimension of a block	64															
Maximum number of threads per block	1024															
Warp size	32															
Maximum number of resident blocks per SM	16		32								16	32	16		24	32
Maximum number of resident warps per SM	64										32	64	48			64
Maximum number of resident threads per SM	2048										1024	2048	1536			2048
Number of 32-bit registers per SM	64 K	128 K	64 K													
Maximum number of 32-bit registers per thread block	64 K				32 K	64 K		32 K	64 K							
Maximum number of 32-bit registers per thread	255															
Maximum amount of shared memory per SM	48 KB	112 KB	64 KB	96 KB	64 KB		96 KB	64 KB	96 KB		64 KB	164 KB	100 KB	164 KB	100 KB	228 KB
Maximum amount of shared memory per thread block [34]	48 KB								96 KB	96 KB	64 KB	163 KB	99 KB	163 KB	99 KB	227 KB
Number of shared memory banks	32															
Maximum amount of local memory per thread	512 KB															
Constant memory size	64 KB															
Cache working set per SM for constant memory	8 KB					4 KB	8 KB									
Cache working set per SM for texture memory	Between 12 KB and 48 KB					Between 24 KB and 48 KB			32 - 128 KB		32 or 64 KB	28KB - 192 KB	28KB - 128 KB	28KB - 192 KB	28KB - 128 KB	28KB - 256 KB
Maximum width for a 1D texture reference bound to a CUDA array	65536					131072										
Maximum width for a 1D texture reference bound to linear memory	2^{27}					2^{28}	2^{27}		2^{28}	2^{27}	2^{28}					
Maximum width and number of layers for a 1D layered texture reference	16384 x 2048					32768 x 2048										
Maximum width and height for a 2D texture reference bound to a CUDA array	65536 x 65536					131072 x 65536										

Task 1

Experiment 1:

Image Dimension: 6000x4000x1

Using **Top Sobel**:

Results:

CPU took: **841137869 ns**
Thus, CPU GFLOPS: 5.42123e+08

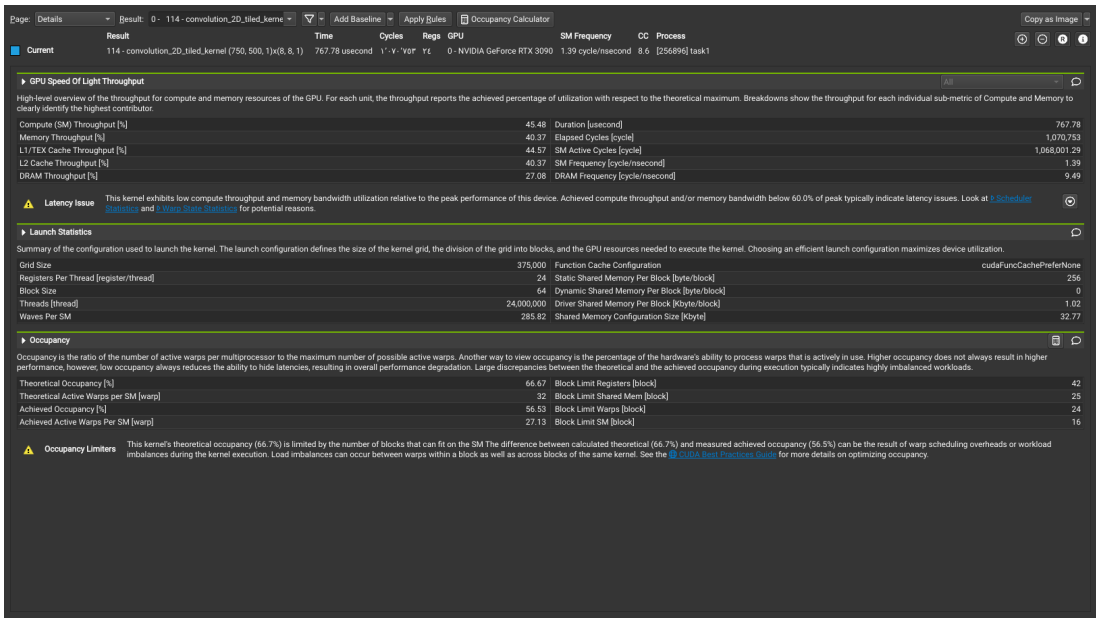
GPU

GPU convolution took **948311** nanoseconds (**WITHOUT DATA TRANSFER**)
GPU convolution took **122397442** nanoseconds (**WITH DATA TRANSFER**)
GPU GFLOPS (without Data Transfer): **4.80855e+11**
GPU GFLOPS(with Data Transfer): **3.72557e+09**

We can see that data transfer is the main bottleneck for slowing the parallel algorithm that much. With Data Transfer, It is faster than the CPU by **6.87**, while GPU without Data Transfer is much faster than the CPU by **886.9**

```
cse-p07-g07f@csep07g07f:~/Ayman and Mina/Again/GPU_programming_with_CUDA/assignment 3/Task1$ nvcc -ljpeg --disable-warnings task1.cu -o task1 && ./task1

        enter a number to choose a convolution kernel:
        1. Blur
        2. Emboss
        3. Outline
        4. Sharpen
        5. Left Sobel
        6. Right Sobel
        7. Top Sobel
        8. Bottom Sobel

image dimensions: 6000x4000x1
image basename: 1.jpg
Applying kernel emboss
CPU time: 841137869 ns
CPU GFLOPS: 5.42123e+08
result image saved to 1.jpg_emboss.jpeg
GPU kernel duration: 948311 ns
GPU total duration: 122397442 ns
GPU GFlops without Data Transfer: 4.80855e+11
GPU GFlops with Data Transfer: 3.72557e+09
gpu result image saved to 1.jpg_emboss_gpu.jpeg
CPU is the same as GPU results
```

GPU Utilization profiling and occupancy analysis: (ZOOM IN for more readability)



Experiment 2:

Results:
Using **Bottom Sobel**:


CPU took: **635346104 ns**
Thus, CPU GFLOPS: 4.81021e+08

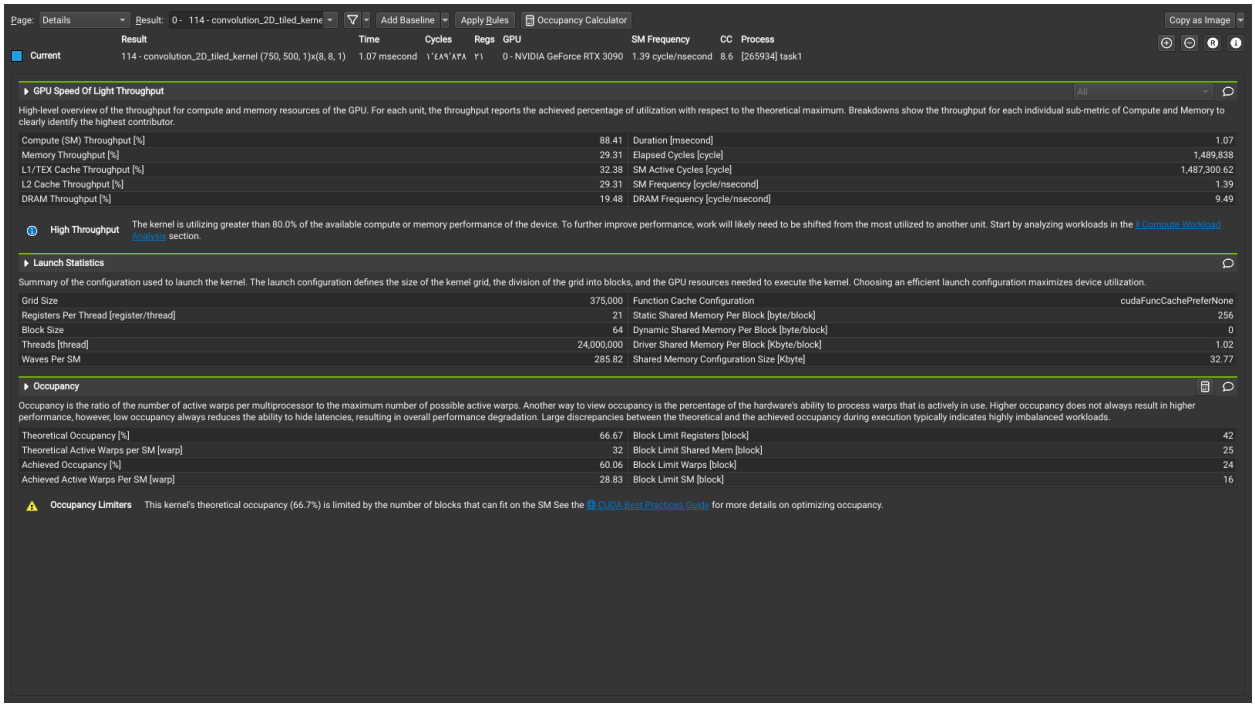
GPU

GPU convolution took **929513** nanoseconds (**WITHOUT DATA TRANSFER**)
GPU convolution took **99401179** nanoseconds (**WITH DATA TRANSFER**)
GPU GFLOPS (without Data Transfer): **3.2879e+11**
GPU GFLOPS(with Data Transfer): **3.07456e+09**

We can see that data transfer is the main bottleneck for slowing the parallel algorithm that much. With Data Transfer, It is faster than the CPU by **6.3**, while GPU without Data Transfer is much faster than the CPU by **683.5**

```
cse-p07-g07f@csep07g07f:~/Ayman and Mina/Again/GPU_programming_with_CUDA/assignment 3/Task1$ nvcc -ljpeg --disable-warnings task1.cu -o task1 && ./task1

        enter a number to choose a convolution kernel:
        1. Blur
        2. Emboss
        3. Outline
        4. Sharpen
        5. Left Sobel
        6. Right Sobel
        7. Top Sobel
        8. Bottom Sobel

image dimensions: 4928x3264x1
image basename: husky.jpeg
Applying kernel bottom_sobel
CPU time: 635346104 ns
CPU GFLOPS: 4.81021e+08
result image saved to husky.jpeg_bottom_sobel.jpeg
GPU kernel duration: 929513 ns
GPU total duration: 99401179 ns
GPU GFlops without Data Transfer: 3.2879e+11
GPU GFlops with Data Transfer: 3.07456e+09
gpu result image saved to husky.jpeg_bottom_sobel_gpu.jpeg
CPU is the same as GPU results

cse-p07-g07f@csep07g07f:~/Ayman and Mina/Again/GPU_programming_with_CUDA/assignment 3/Task1$
```

GPU Utilization profiling and occupancy analysis: (ZOOM IN for more readability)



Experiment 3: Outline

CPU took: **2695977 ns**
Thus, CPU GFLOPS: 4.61867e+08

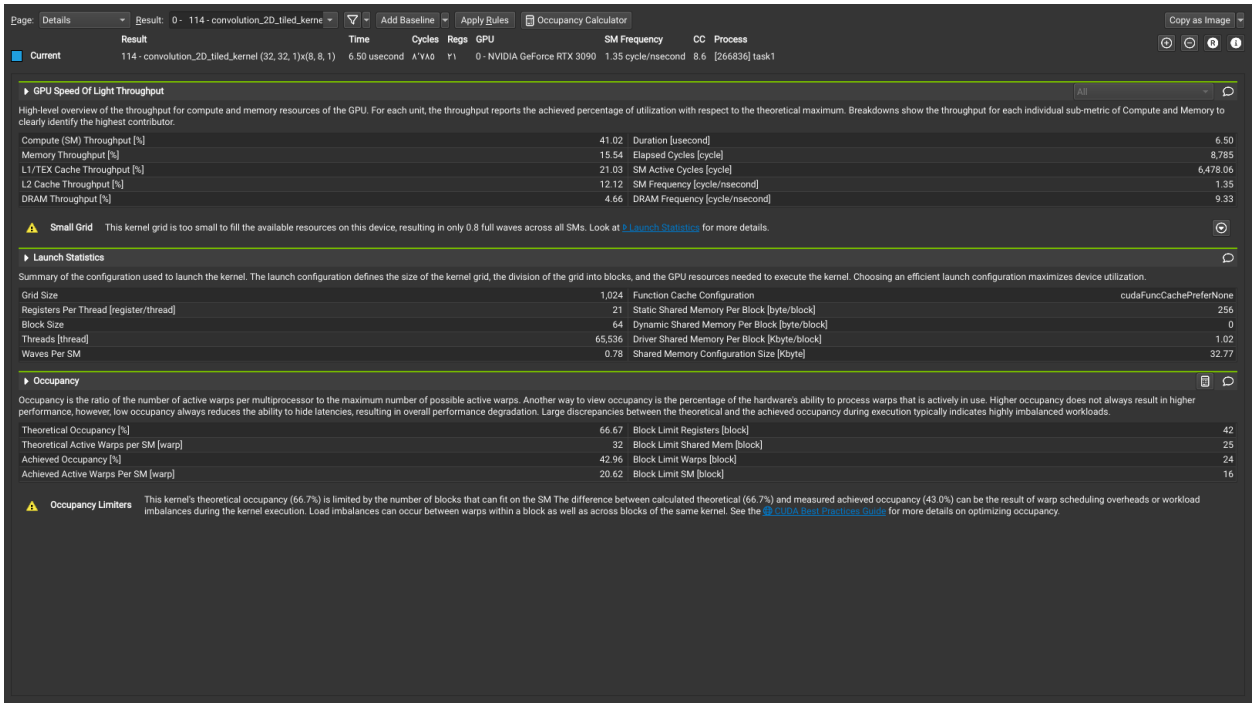
GPU

GPU convolution took **24940** nanoseconds (**WITHOUT DATA TRANSFER**)
GPU convolution took **85843748** nanoseconds (**WITH DATA TRANSFER**)
GPU GFLOPS (without Data Transfer): **4.99272e+10**
GPU GFLOPS(with Data Transfer): **1.45052e+07**

We can see that data transfer is the main bottleneck for slowing the parallel algorithm that much. With Data Transfer, CPU is faster than GPU with data transfer, while GPU without Data Transfer is much faster than the CPU by **108.5. Compared** to other experiments, we can see that GPU is much faster with larger images.

```
cse-p07-g07f@csep07g07f:~/Ayman and Mina/Again/GPU_programming_with_CUDA/assignment 3/Task1$ nvcc -ljpeg --disable-warnings task1.cu -o task1 && ./task1

        enter a number to choose a convolution kernel:
        1. Blur
        2. Emboss
        3. Outline
        4. Sharpen
        5. Left Sobel
        6. Right Sobel
        7. Top Sobel
        8. Bottom Sobel

image dimensions: 256x256x1
image basename: lena.jpeg
Applying kernel outline
CPU time: 2695977 ns
CPU GFLOPS: 4.61867e+08
result image saved to lena.jpeg_outline.jpeg
GPU kernel duration: 24940 ns
GPU total duration: 85843748 ns
GPU GFlops without Data Transfer: 4.99272e+10
GPU GFlops with Data Transfer: 1.45052e+07
gpu result image saved to lena.jpeg_outline_gpu.jpeg
CPU is the same as GPU results

cse-p07-g07f@csep07g07f:~/Ayman and Mina/Again/GPU_programming_with_CUDA/assignment 3/Task1$
Live Share                                   You, 1 second ago  Ln 347, Col 22  Spaces: 4
```

GPU Utilization profiling and occupancy analysis: (ZOOM IN for more readability)



Task 2

Experiment 1:

Using 6000x4000 Image size

CPU took: **0.100483 seconds (100483205 ns)**
Thus, CPU GFLOPS: 9.52928e+08

GPU

Total Number Of Blocks**: 6 * 4000 + 4 * 6000 = 48000**
Total number of threads: **1024 * 6 * 4000 + 4*1024 *6000 = 49152000**
GPU 2D-PrefixSum took **4719036** nanoseconds (**WITHOUT DATA TRANSFER**)
GPU 2D-PrefixSum took **150374009** nanoseconds (**WITH DATA TRANSFER**)
GPU GFLOPS (without Data Transfer): **1.45052e+11**
GPU GFLOPS(with Data Transfer): **4.52022e+09**

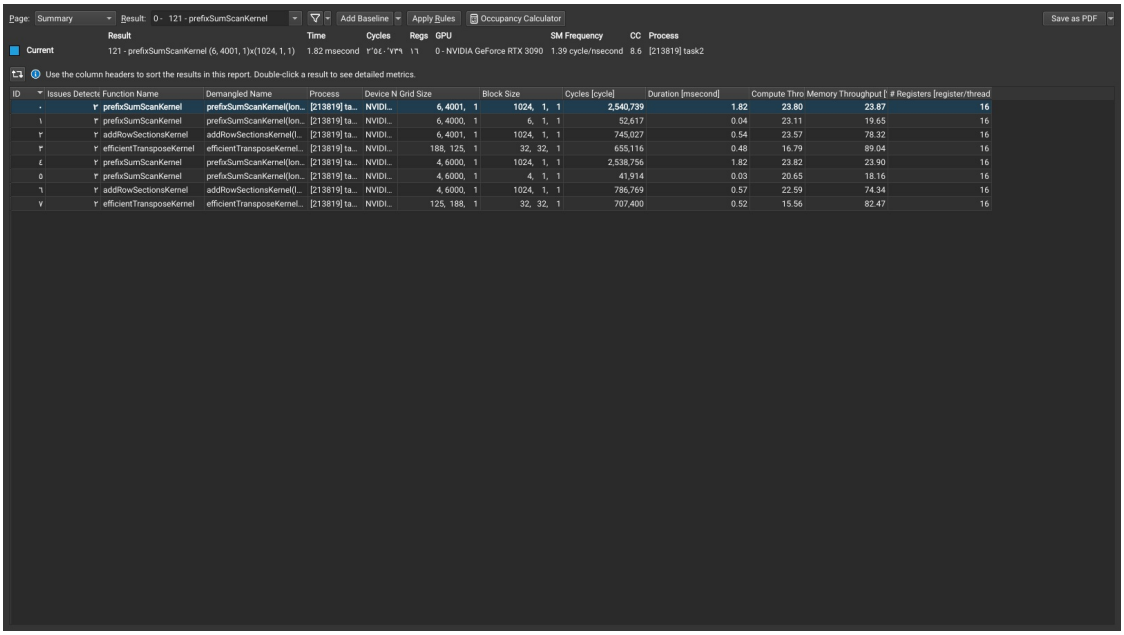
We can see that data transfer is the main bottleneck for slowing the parallel algorithm that much. With Data Transfer, It is faster than the CPU by **4.8**, while GPU without Data Transfer is much faster than the CPU by **76.4**

```
cse-p07-g07f@csep07g07f:~/Ayman and Mina/Again/GPU_programming_with_CUDA/assignment 3/Task2$ nvcc -ljpeg --disable-warnings task2.cu -o task2 && ./task2
5 2 5 2
3 6 3 6
5 2 5 2
3 6 3 6

Results:

5 7 12 14
8 16 24 32
13 23 36 46
16 32 48 64

Test passed
Test passed!!
image dimensions: 6000x4000x1
image basename: 1.jpg
CPU GFLOPS: 9.52928e+08
CPU duration: 100742149 ns
The Genralized Kernel of GPU
GPU duration without Data Transfer: 4709688 ns
GPU duration with Data Transfer: 150622138 ns
GPU GFLOPS Without Transfer: 1.4534e+11
GPU GFLOPS with Transfer: 4.54453e+09
Comparing 24000000 elements
Test passed
```

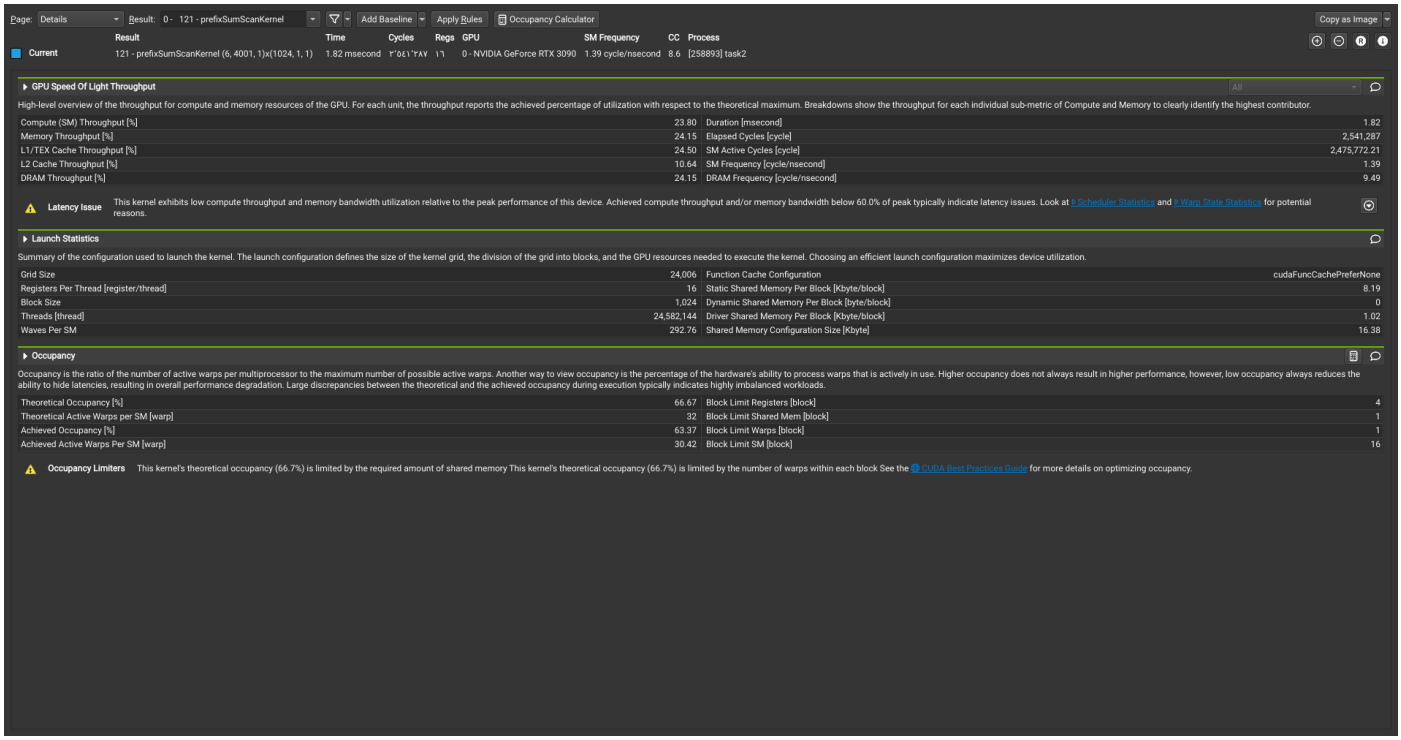
Test Query functionality:

```
Test passed
The GPU Test Passed
Enter the number of queries to execute :
1
Enter query 0 :
Enter A(x1,y1)
0 0
Enter B(x2,y2)
1 0
Enter C(x3,y3)
0 1
Enter D(x4,y4)
1 1
Result: 11
cse-p07-g07f@csep07g07f:~/Ayman and Mina/Again/GPU_programming_with_CUDA/assignment 3/Task2$
Live Share
```

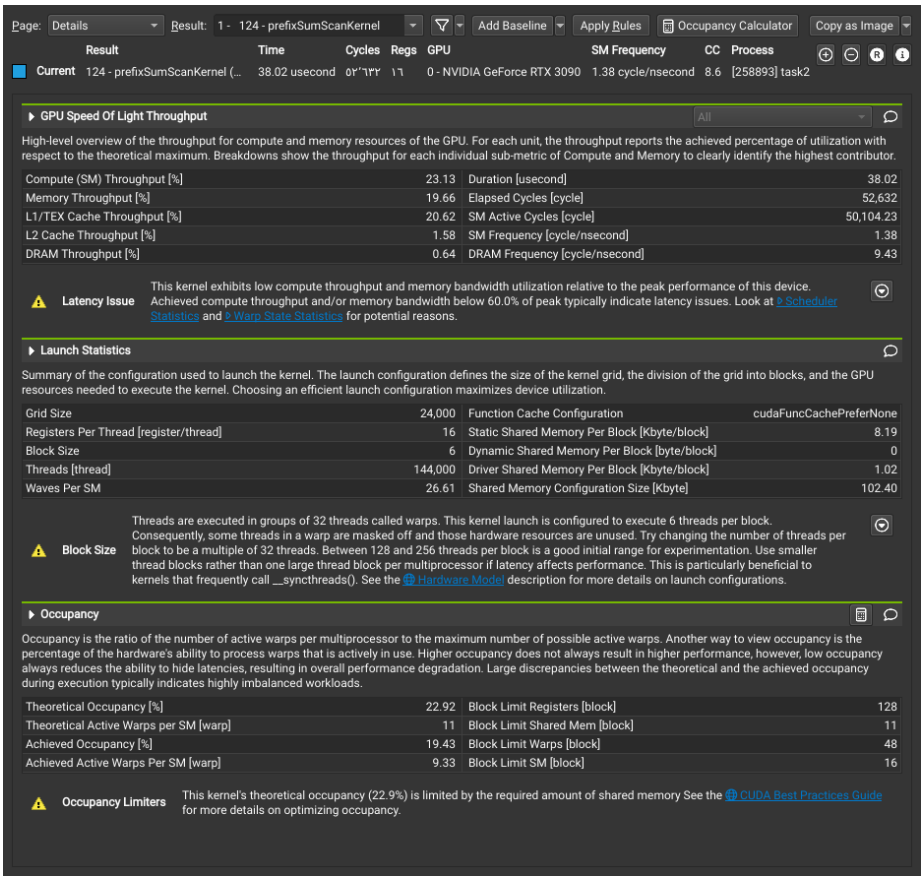
GPU Utilization profiling and occupancy analysis: (ZOOM IN for more readability):
8 kernels: 1 for row, 1, col, 2x prefix_sum to section, 2x propagate, 2x transpose.



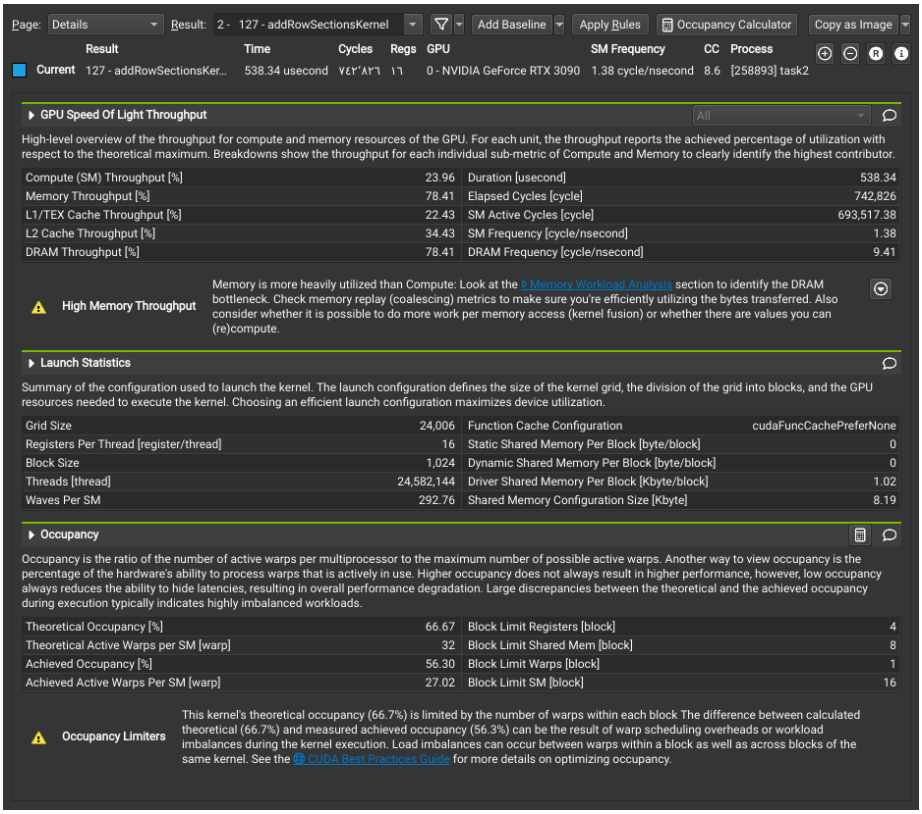
First Kernel:



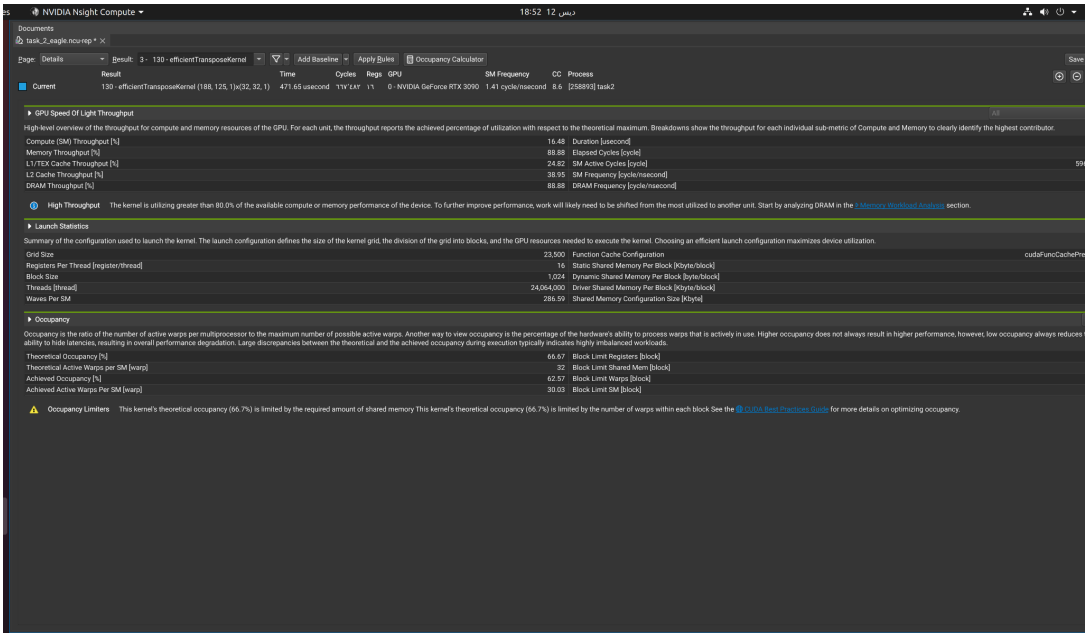
Second Kernel:



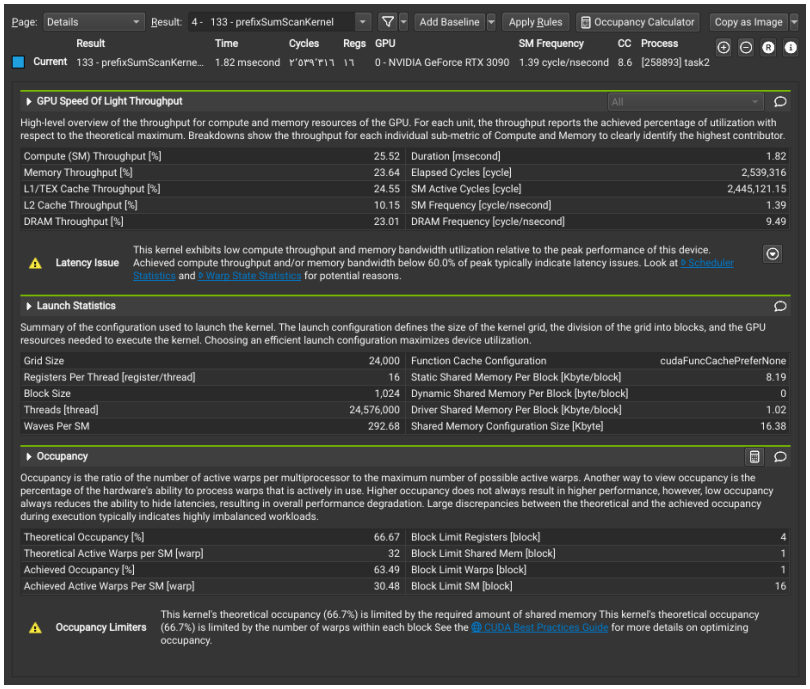
Third Kernel:



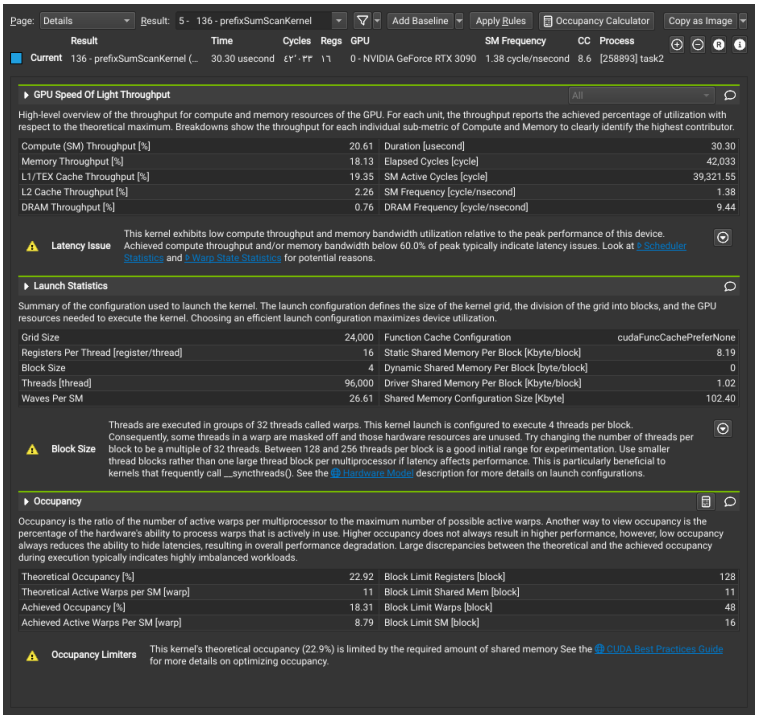
Fourth Kernel:



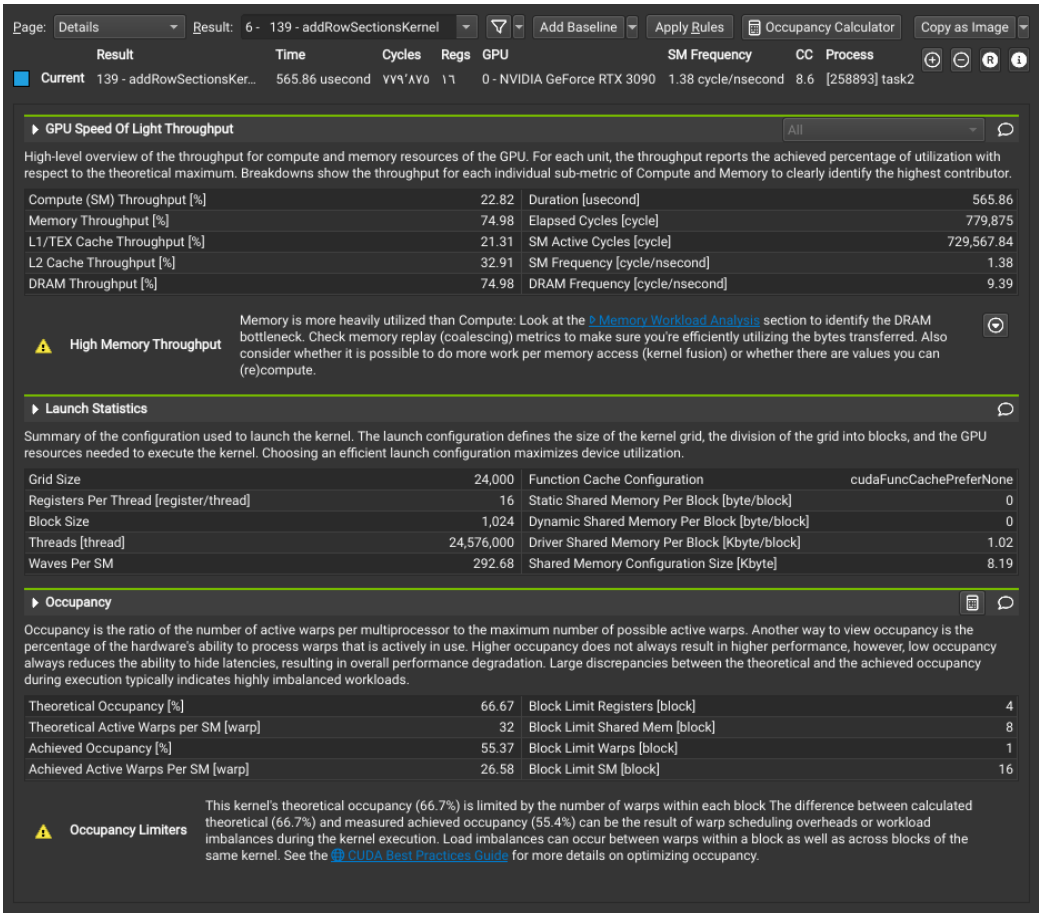
Fifth Kernel:



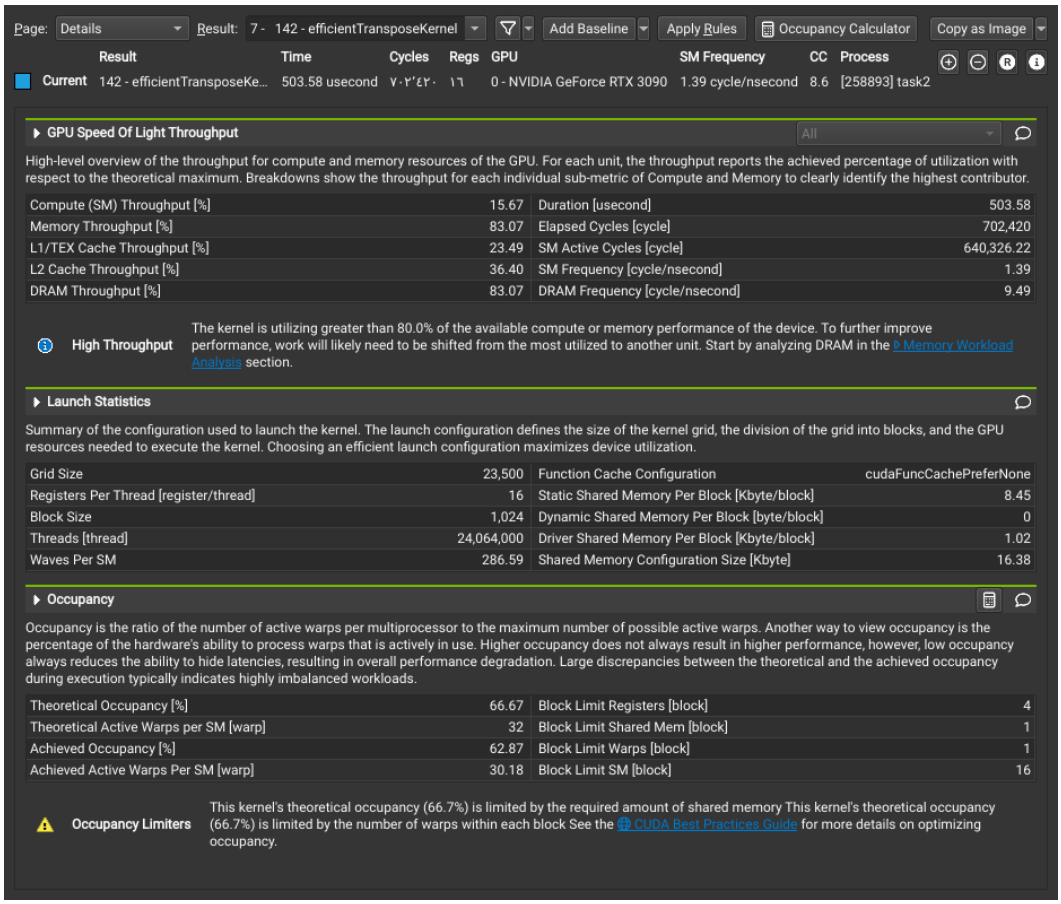
Sixth Kernel:



Seventh Kernel:



Eighth Kernel:



Experiment 2:

Using 4000x2667 image size

CPU took: **47088192 ns**
Thus, CPU GFLOPS: 9.06214e+08

GPU

GPU 2D-PrefixSum took **3334478** nanoseconds (**WITHOUT DATA TRANSFER**)
GPU 2D-PrefixSum took **117054717** nanoseconds (**WITH DATA TRANSFER**)
GPU GFLOPS (without Data Transfer): **8.75027e+10**
GPU GFLOPS(with Data Transfer): **2.49264e+09**

We can see that data transfer is the main bottleneck for slowing the parallel algorithm that much. With Data Transfer, It is faster than the CPU by **2.8**, while GPU without Data Transfer is much faster than the CPU by **96.6**

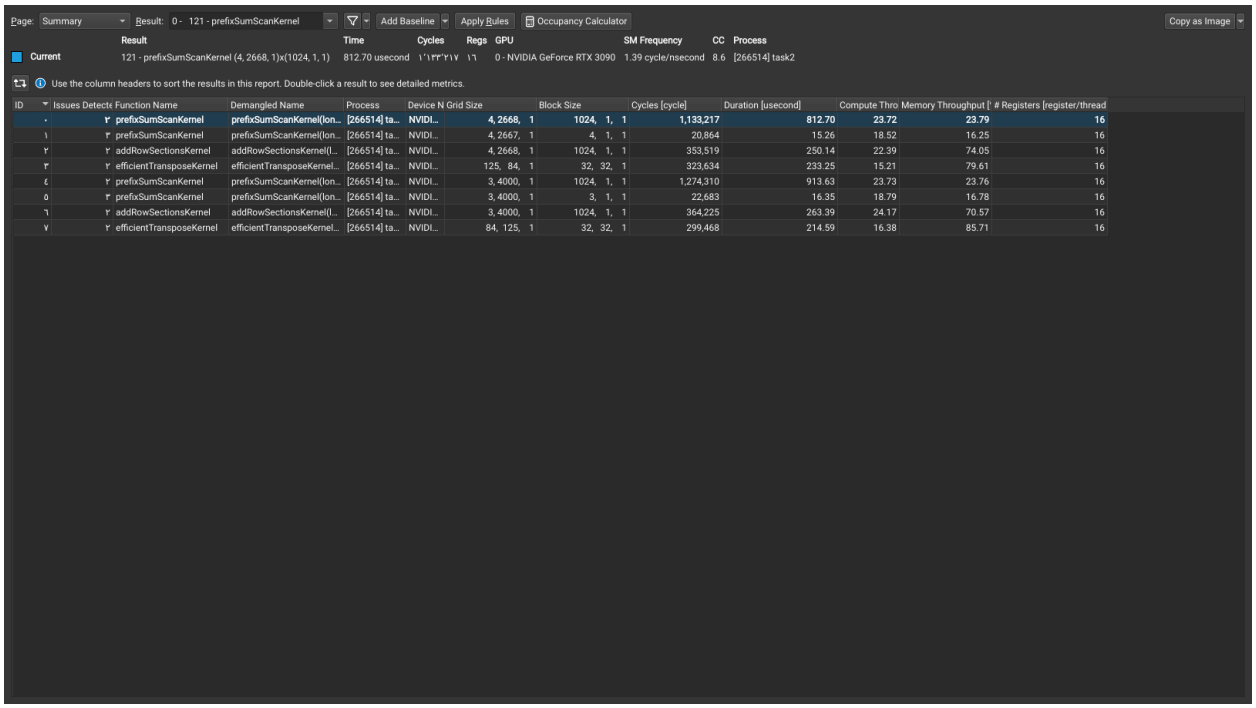
```
Test passed
Test passed!!
image dimensions: 4000x2667x1
image basename: greek.jpg
CPU GFLOPS: 9.06214e+08
CPU duration: 47088192 ns
The Genralized Kernel of GPU
GPU duration without Data Transfer: 3334478 ns
GPU duration with Data Transfer: 117054717 ns
GPU GFLOPS Without Transfer: 8.75027e+10
GPU GFLOPS with Transfer: 2.49264e+09
Comparing 10668000 elements
Test passed
The GPU Test Passed
cse-p07-g07f@csep07g07f:~/Ayman and Mina/Again/GPU_programming_with_CUDA/assignment 3/Task2$
```

Test Query functionality:

```
The GPU Test Passed
Enter the number of queries to execute :
1
Enter query 0 :
Enter A(x1,y1)
5 5
Enter B(x2,y2)
100 5
Enter C(x3,y3)
5 100
Enter D(x4,y4)
100 100
Result: 1037511
cse-p07-g07f@csep07g07f:~/Ayman and Mina/Again/GPU_programming_with_CUDA/assignment 3/Task2$
Live Share
```

GPU Utilization profiling and occupancy analysis: (ZOOM IN for more readability):

I will add just an overview here.



Task 3

Experiment 1: with 255 bins

834x390x3 image Size

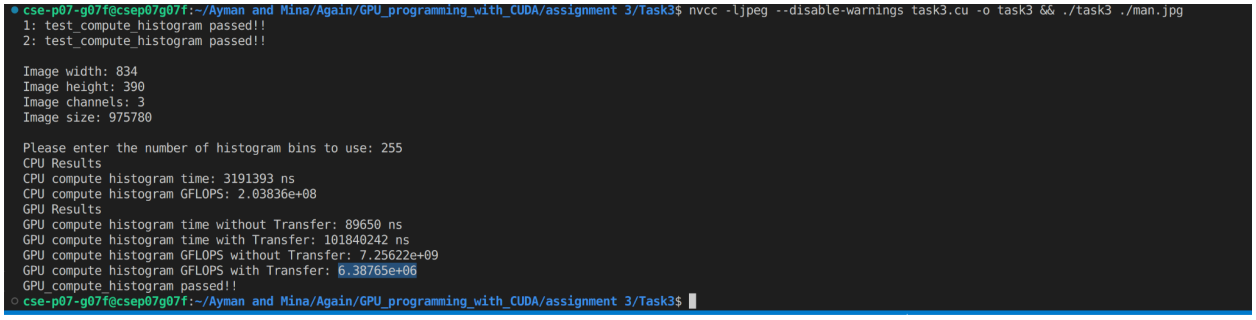
Results:

CPU took: **3191393 ns**
Thus, CPU GFLOPS: 2.03836e+08

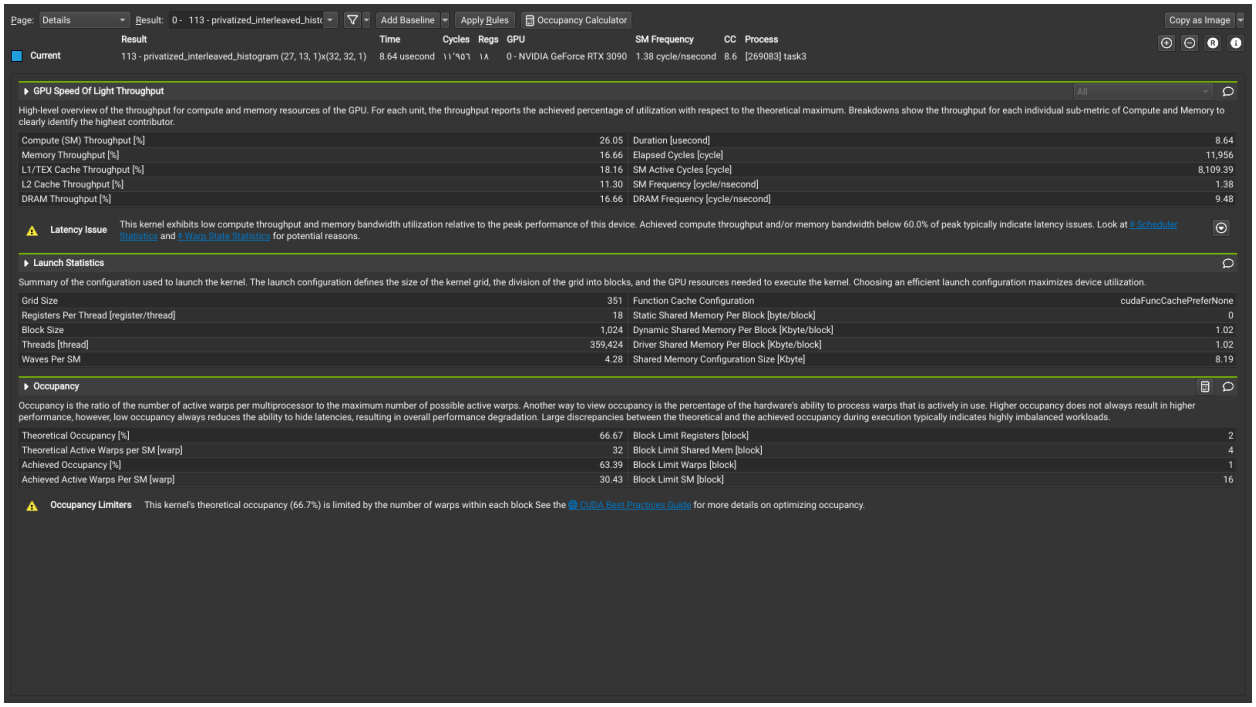
GPU

GPU convolution took **89650** nanoseconds (**WITHOUT DATA TRANSFER**)
GPU convolution took **101840242** nanoseconds (**WITH DATA TRANSFER**)
GPU GFLOPS (without Data Transfer): **7.25622e+09**
GPU GFLOPS(with Data Transfer): **6.38765e+06**

We can see that data transfer is the main bottleneck for slowing the parallel algorithm that much. With Data Transfer, It is not faster than CPU, as the image is very small. However, GPU without Data Transfer is much faster than the CPU by **35.7**



GPU Utilization profiling and occupancy analysis: (ZOOM IN for more readability):



Experiment 2: with 256 bins

615x418x3 image Size

CPU took: **555549 ns**
Thus, CPU GFLOPS: 9.25463e+08

GPU

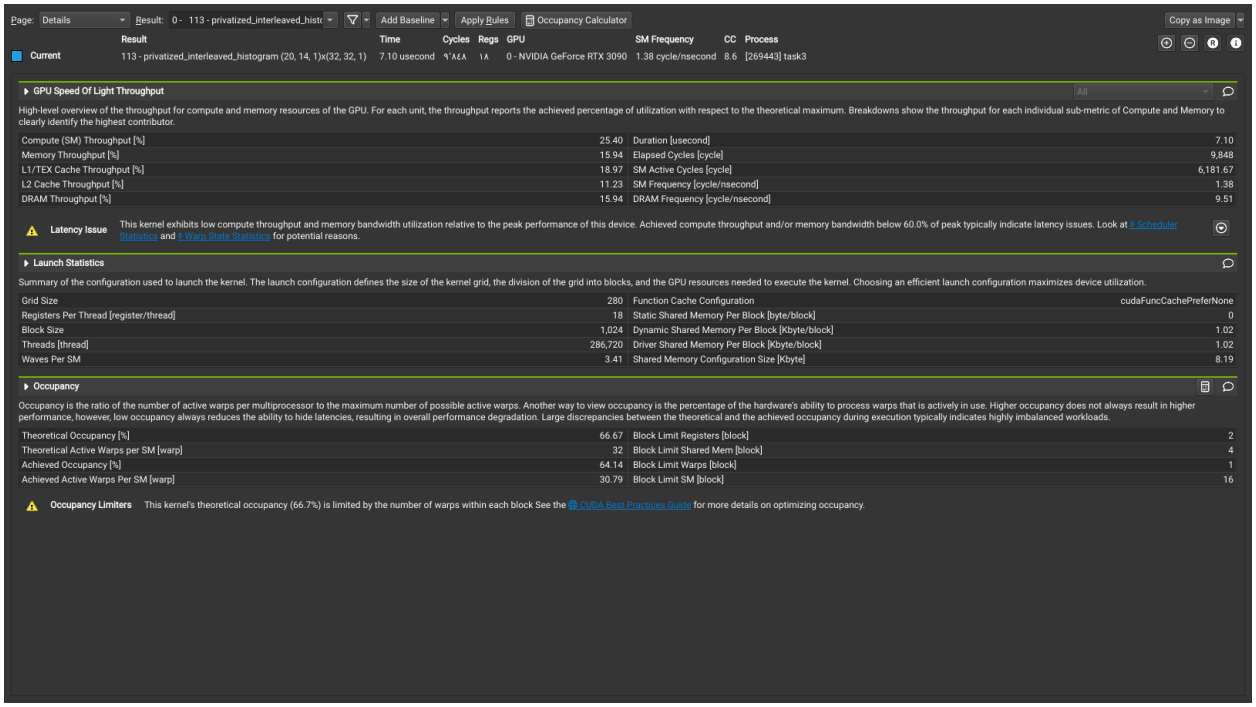
GPU convolution took **87684** nanoseconds (**WITHOUT DATA TRANSFER**)
GPU convolution took **90681306** nanoseconds (**WITH DATA TRANSFER**)

GPU GFLOPS (without Data Transfer): **5.86356e+09**
GPU GFLOPS(with Data Transfer): **5.66975e+06**

We can see that data transfer is the main bottleneck for slowing the parallel algorithm that much. With Data Transfer, It is not faster than CPU, as the image is very small. However, GPU without Data Transfer is much faster than the CPU by **35.7**

```
cse-p07-g07f@csep07g07f:~/Ayman and Mina/Again/GPU_programming_with_CUDA/assignment 3/Task3$ nvcc -ljpeg --disable-warnings task3.cu -o task3 && ./task3
1: test_compute_histogram passed!!
2: test_compute_histogram passed!!

Image width: 615
Image height: 418
Image channels: 3
Image size: 771210

Please enter the number of histogram bins to use: CPU Results
CPU compute histogram time: 555549 ns
CPU compute histogram GFLOPS: 9.25463e+08
GPU Results
GPU compute histogram time without Transfer: 87684 ns
GPU compute histogram time with Transfer: 90681306 ns
GPU compute histogram GFLOPS without Transfer: 5.86356e+09
GPU compute histogram GFLOPS with Transfer: 5.66975e+06
GPU_compute_histogram passed!!
cse-p07-g07f@csep07g07f:~/Ayman and Mina/Again/GPU_programming_with_CUDA/assignment 3/Task3$
```

GPU Utilization profiling and occupancy analysis: (ZOOM IN for more readability):



Experiment 3: 150 bins

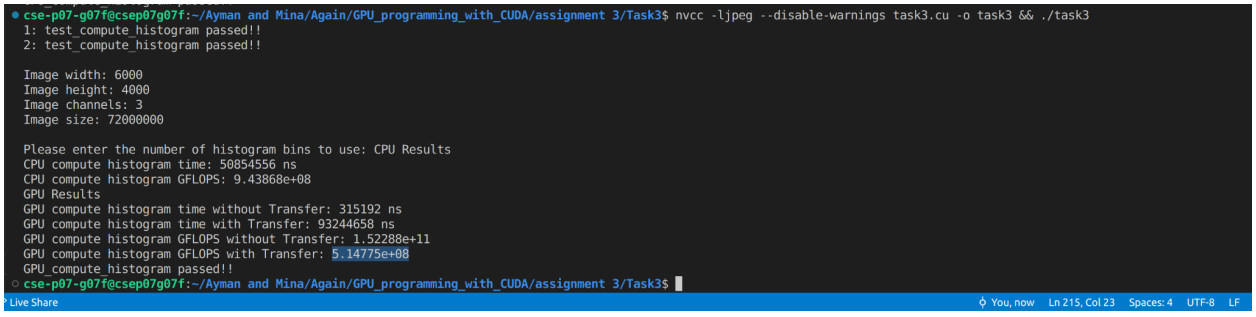
6000x4000x3 image Size

CPU took: **50854556 ns**
Thus, CPU GFLOPS: 9.43868e+08

GPU

GPU convolution took **315192** nanoseconds (**WITHOUT DATA TRANSFER**)
GPU convolution took **93244658** nanoseconds (**WITH DATA TRANSFER**)
GPU GFLOPS (without Data Transfer): **1.52288e+11**
GPU GFLOPS(with Data Transfer): **5.14775e+08**

We can see that data transfer is the main bottleneck for slowing the parallel algorithm that much. With Data Transfer, It is not faster than CPU, as the image is very small. However, GPU without Data Transfer is much faster than the CPU by **161.34**



GPU Utilization profiling and occupancy analysis: (ZOOM IN for more readability):