Siddharth Nair
2017196

Assignment 2

Part 1:

1. An array was created to store the minimum distance for all the pixels.
2. 2 kernels were defined. One for calculating the minimum distance and one for getting the final signed distance transform.
3. The number of edge pixels was calculated on the host as it is a serial operation.
4. The edge pixel array was split in 20 parts and the kernel for calculating the minimum distance was called twenty times. (This was done because the shared memory is unable to hold the entire edge pixel array for the larger image).
5. The second kernel is then called to calculate the signed distance transform using the array created in step 1.

	CPU	GPU (overall)	GPU (kernel)	MSE
Image 1 (tree8_small512.png)	1.827 s	0.0156 s	0.0118 s	0
Image 2 (tree8.png)	279.58 s	1.947 s	1.688 s	0

Part 2:

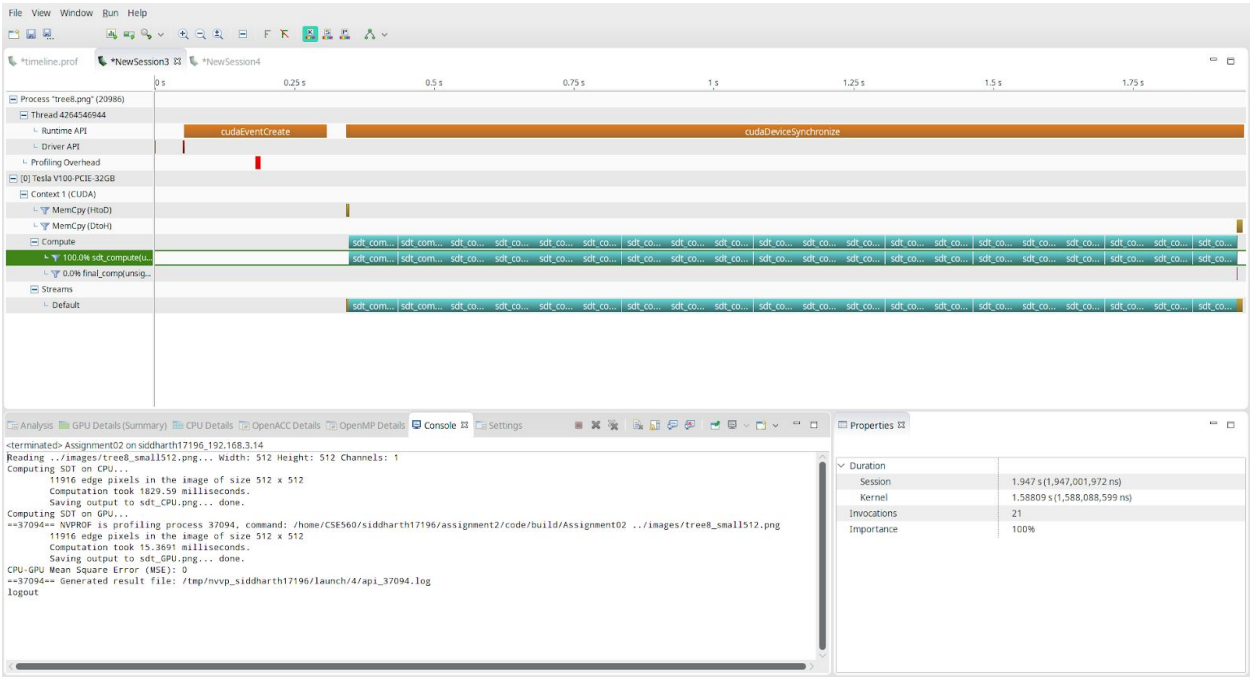
Constant memory instead of shared memory is a good choice because in that case the number of “read’s” from the global memory into the shared memory would be reduced. Since the global memory access is slower than that of the constant memory hence using constant memory would result in some speedup.

Part 3:

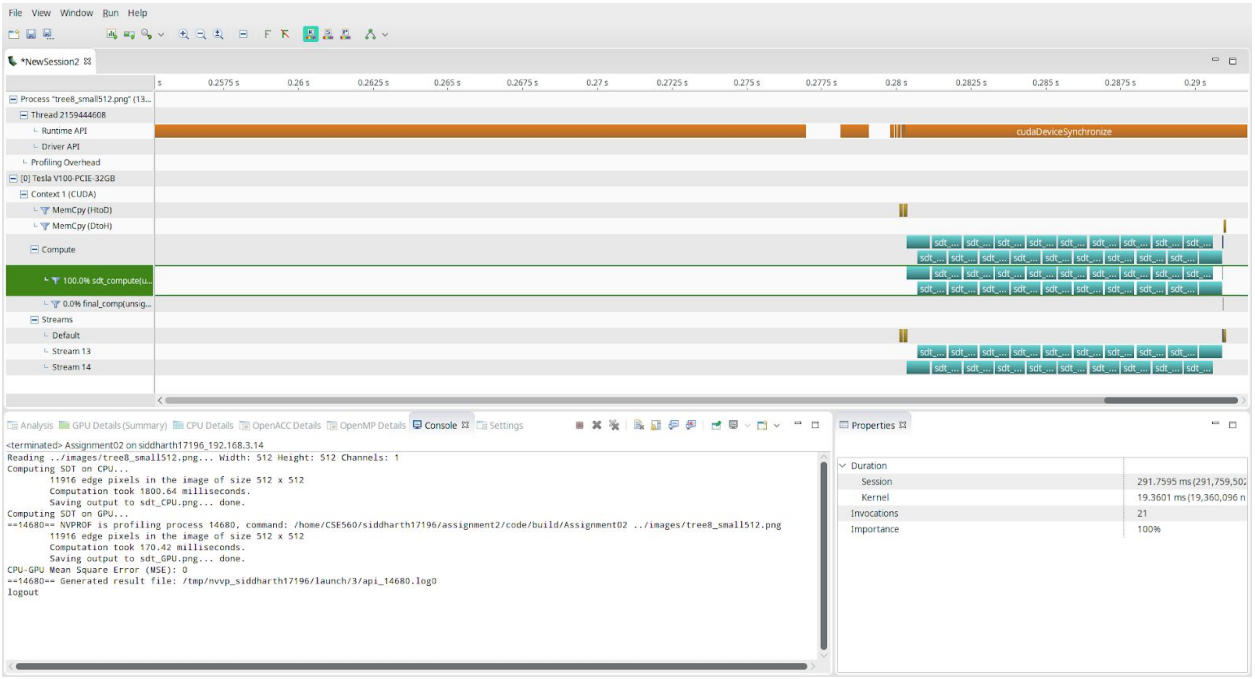
Achieved Occupancy:

- Smaller image - 0.834
- Larger image - 0.516

As the selected row below indicates, most of the time is spent in the kernel calls. Since there is no kernel level concurrency all the kernels are executing serially. The time could be further reduced by making the kernels concurrent.



Part 4:



In this version of the code, I have used two streams to achieve concurrency.

1. In this case 2 arrays were created to calculate the minimum distances for each pixel from both the streams concurrently
2. The second kernel was called at the end to select the minimum from the two arrays and calculate the signed distance transform.

	Before			After		
	GPU (overall)	GPU (kernel)	Occupancy	GPU(overall)	GPU (kernel)	Occupancy
Image 1 (tree8_small512.png)	0.0156 s	0.0118 s	0.834	0.0162 s	0.0098 s	0.834
Image 2 (tree8.png)	1.947 s	1.688 s	0.516	1.911 s	1.5 s	0.736

NOTE: In the folder a cuda file has been attached *(kernel_part1.cu*). This was the cuda code for the first part (shared memory only without any optimisations).

The main executable has however been made using the final code.