The process of obtaining a new image was performed by applying an affine transformation matrix randomly generated to the image given in this section.

Make sure you have set up OpenCV and NumPy for use in serial and parallel implementation, and CUDA for use in parallel implementation.

Sample images you can use are given in the dataset folder. The outputs obtained at the end of the program are saved in the output file.

You can use the following command to run the affineDeformation.py file:

**python3 affineDeformation.py -f fileName.xxx**

*Note: fileName.xxx is the input image for which you want to apply the affine transformation process.*

When you run it without any problems, you should get an output as follows:

```
(base) derya@derya:~/Desktop/affineDeformation/dataset$ python3 affineDeformation.py -f 3.pgm
->Loading images
        3.pgm successfully read!
Runtime of the program is 0.0056607723236083984
(base) derya@derya:~/Desktop/affineDeformation/dataset$
```

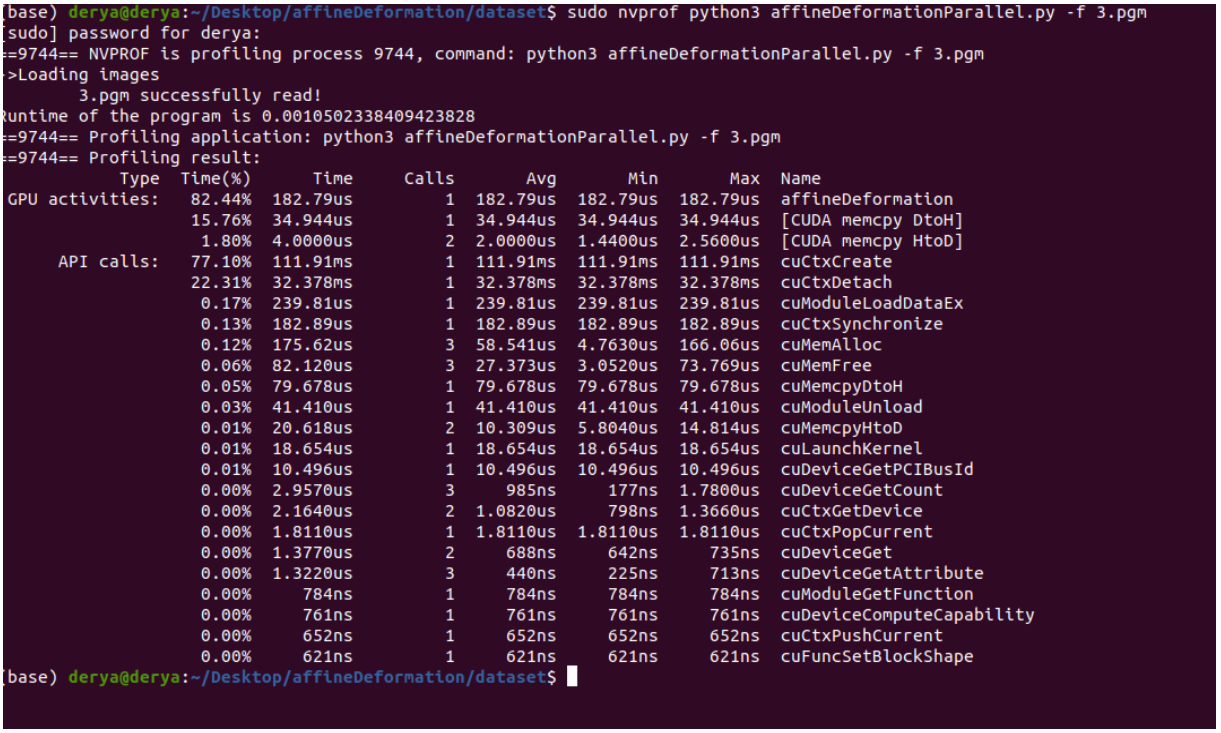
You can use the following command to run the affineDeformationParallel.py file:

**sudo nvprof python3 affineDeformationParallel.py -f fileName.xxx**

In this part, If you get an error regarding the installation of pycuda while running this command, you can run the following command for installation:

**sudo pip install pycuda**

When you run it without any problems, you should get an output as follows:

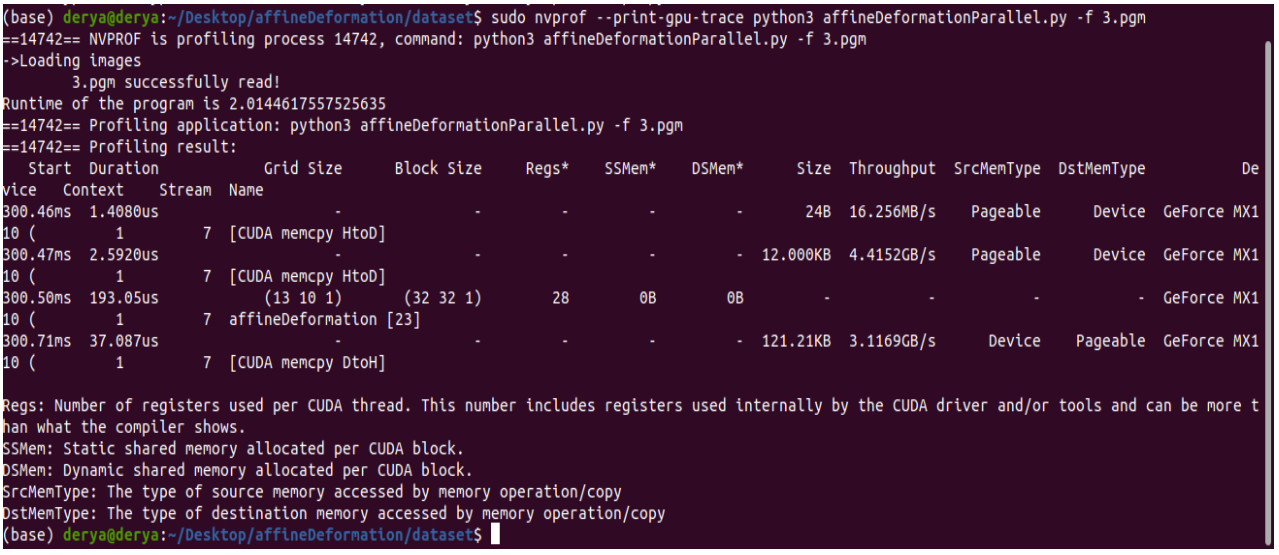


```
(base) derya@derya:~/Desktop/affineDeformation/dataset$ sudo nvprof python3 affineDeformationParallel.py -f 3.pgm
[sudo] password for derya:
==9744== NVPROF is profiling process 9744, command: python3 affineDeformationParallel.py -f 3.pgm
->Loading images
       3.pgm successfully read!
Runtime of the program is 0.0010502338409423828
==9744== Profiling application: python3 affineDeformationParallel.py -f 3.pgm
==9744== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   82.44%  182.79us         1  182.79us  182.79us  182.79us  affineDeformation
                   15.76%  34.944us         1  34.944us  34.944us  34.944us  [CUDA memcpy DtoH]
                    1.80%  4.0000us         2  2.0000us  1.4400us  2.5600us  [CUDA memcpy HtoD]
      API calls:   77.10%  111.91ms         1  111.91ms  111.91ms  111.91ms  cuCtxCreate
                   22.31%  32.378ms         1  32.378ms  32.378ms  32.378ms  cuCtxDetach
                    0.17%  239.81us         1  239.81us  239.81us  239.81us  cuModuleLoadDataEx
                    0.13%  182.89us         1  182.89us  182.89us  182.89us  cuCtxSynchronize
                    0.12%  175.62us         3  58.541us  4.7630us  166.06us  cuMemAlloc
                    0.06%  82.120us         3  27.373us  3.0520us  73.769us  cuMemFree
                    0.05%  79.678us         1  79.678us  79.678us  79.678us  cuMemcpyDtoH
                    0.03%  41.410us         1  41.410us  41.410us  41.410us  cuModuleUnload
                    0.01%  20.618us         2  10.309us  5.8040us  14.814us  cuMemcpyHtoD
                    0.01%  18.654us         1  18.654us  18.654us  18.654us  cuLaunchKernel
                    0.01%  10.496us         1  10.496us  10.496us  10.496us  cuDeviceGetPCIBusId
                    0.00%  2.9570us         3     985ns     177ns  1.7800us  cuDeviceGetCount
                    0.00%  2.1640us         2  1.0820us     798ns  1.3660us  cuCtxGetDevice
                    0.00%  1.8110us         1  1.8110us  1.8110us  1.8110us  cuCtxPopCurrent
                    0.00%  1.3770us         2     688ns     642ns     735ns  cuDeviceGet
                    0.00%  1.3220us         3     440ns     225ns     713ns  cuDeviceGetAttribute
                    0.00%     784ns         1     784ns     784ns     784ns  cuModuleGetFunction
                    0.00%     761ns         1     761ns     761ns     761ns  cuDeviceComputeCapability
                    0.00%     652ns         1     652ns     652ns     652ns  cuCtxPushCurrent
                    0.00%     621ns         1     621ns     621ns     621ns  cuFuncSetBlockShape
(base) derya@derya:~/Desktop/affineDeformation/dataset$
```

You can also use --print-gpu-trace. You can use the following command by running:

**sudo nvprof --print-gpu-trace python3 affineDeformationParallel.py -f fileName.xxx**

When you run it without any problems, you should get an output as follows:



```
(base) derya@derya:~/Desktop/affineDeformation/dataset$ sudo nvprof --print-gpu-trace python3 affineDeformationParallel.py -f 3.pgm
==14742== NVPROF is profiling process 14742, command: python3 affineDeformationParallel.py -f 3.pgm
->Loading images
       3.pgm successfully read!
Runtime of the program is 2.0144617557525635
==14742== Profiling application: python3 affineDeformationParallel.py -f 3.pgm
==14742== Profiling result:
   Start  Duration           Grid Size      Block Size     Regs*    SSMem*    DSMem*      Size  Throughput  SrcMemType  DstMemType           De
vice   Context    Stream  Name
300.46ms  1.4080us                    -               -         -         -         -      24B  16.256MB/s    Pageable      Device  GeForce MX1
10 (         1         7  [CUDA memcpy HtoD]
300.47ms  2.5920us                    -               -         -         -         -  12.000KB  4.4152GB/s    Pageable      Device  GeForce MX1
10 (         1         7  [CUDA memcpy HtoD]
300.50ms  193.05us           (13 10 1)      (32 32 1)        28        0B        0B         -           -           -           -  GeForce MX1
10 (         1         7  affineDeformation [23]
300.71ms  37.087us                    -               -         -         -         -  121.21KB  3.1169GB/s      Device    Pageable  GeForce MX1
10 (         1         7  [CUDA memcpy DtoH]

Regs: Number of registers used per CUDA thread. This number includes registers used internally by the CUDA driver and/or tools and can be more t
han what the compiler shows.
SSMem: Static shared memory allocated per CUDA block.
DSMem: Dynamic shared memory allocated per CUDA block.
SrcMemType: The type of source memory accessed by memory operation/copy
DstMemType: The type of destination memory accessed by memory operation/copy
(base) derya@derya:~/Desktop/affineDeformation/dataset$
```

If you do not want a detailed output like the one above, you can run the following command:

**python3 affineDeformationParallel.py -f fileName.xxx**