10 Runtimes from DARWIN

All runtimes presented are ~1000 times faster than the serial program

Problem Size (Width x Height)	Number of Frames	CUDA Compute Time
1024 x 768	10	0.008289
1280 x 720	20	0.015956
1600 x 900	30	0.034725
2048 x 1080	15	0.027261
1920 x 1080	40	0.064739
1920 x 1200	45	0.086726
2560 x 1440	50	0.143293
3200 x 1800	60	0.275448
3840 x 2160	30	0.201792
4096 x 2160	60	0.419613

Bonus Question (fractal_float.cu)

```
clawsonx@cisc372:~/Assignment4$ srun ./fractal_float_cuda 2048 1080 60
Fractal v1.6 [CUDA] - Float Version
Computing 60 frames of 2048 by 1080 fractal
CUDA compute time: 0.205700 s
clawsonx@cisc372:~/Assignment4$ srun ./fractal_cuda 2048 1080 60
Fractal v1.6 [CUDA]
Computing 60 frames of 2048 by 1080 fractal
CUDA compute time: 0.387408 s
clawsonx@cisc372:~/Assignment4$ srun ./fractal_float_cuda 1280 720 30
Fractal v1.6 [CUDA] - Float Version
Computing 30 frames of 1280 by 720 fractal
CUDA compute time: 0.045442 s
clawsonx@cisc372:~/Assignment4$ srun ./fractal_cuda 1280 720 30
Fractal v1.6 [CUDA]
Computing 30 frames of 1280 by 720 fractal
CUDA compute time: 0.084795 s
clawsonx@cisc372:~/Assignment4$ srun ./fractal_float_cuda 1920 1200 45
Fractal v1.6 [CUDA] - Float Version
Computing 45 frames of 1920 by 1200 fractal
CUDA compute time: 0.162947 s
clawsonx@cisc372:~/Assignment4$ srun ./fractal_cuda 1920 1200 45
Fractal v1.6 [CUDA]
Computing 45 frames of 1920 by 1200 fractal
CUDA compute time: 0.305769 s
clawsonx@cisc372:~/Assignment4$
```

Bonus Question Observations

The runtimes between ‘fractal.cu’ and ‘fractal_float.cu’ are expected. This is because float uses half the memory bandwidth of double. On CUDA devices, using float instead of double reduces memory transfers, which can lead to performance improvements. The results show fractal.cu taking just about twice as long as fractal_float.cu checks out in this situation.

Float values also typically use fewer registers on the GPU. This allows for more registers to be available for other computations or for handling a higher number of threads. This can result in more efficient execution and faster kernel performance.