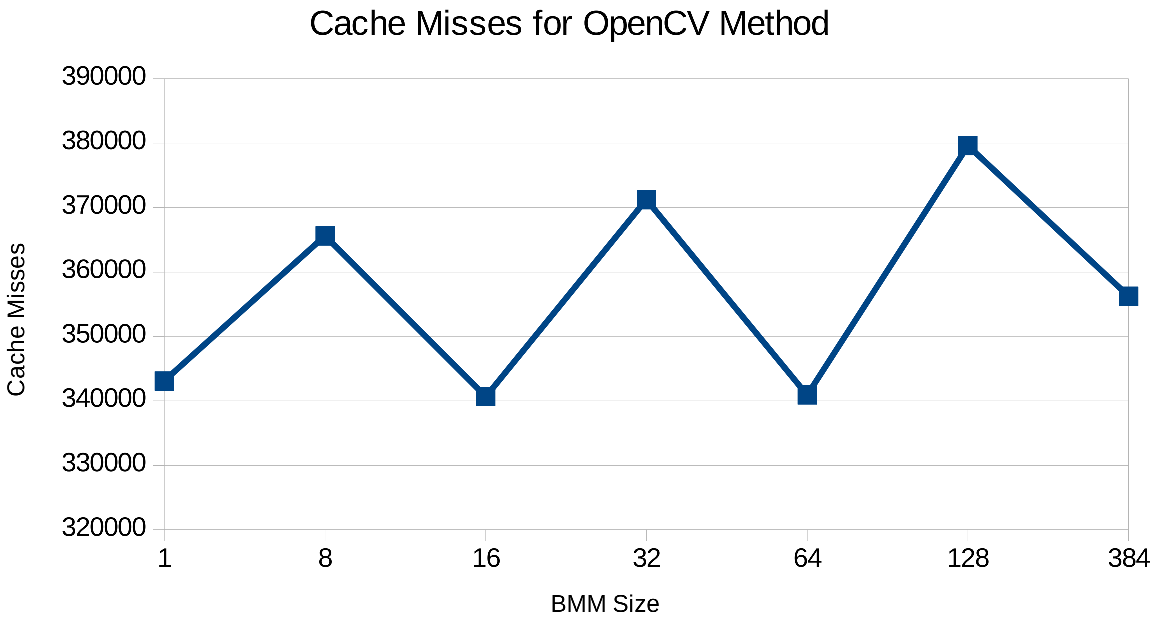
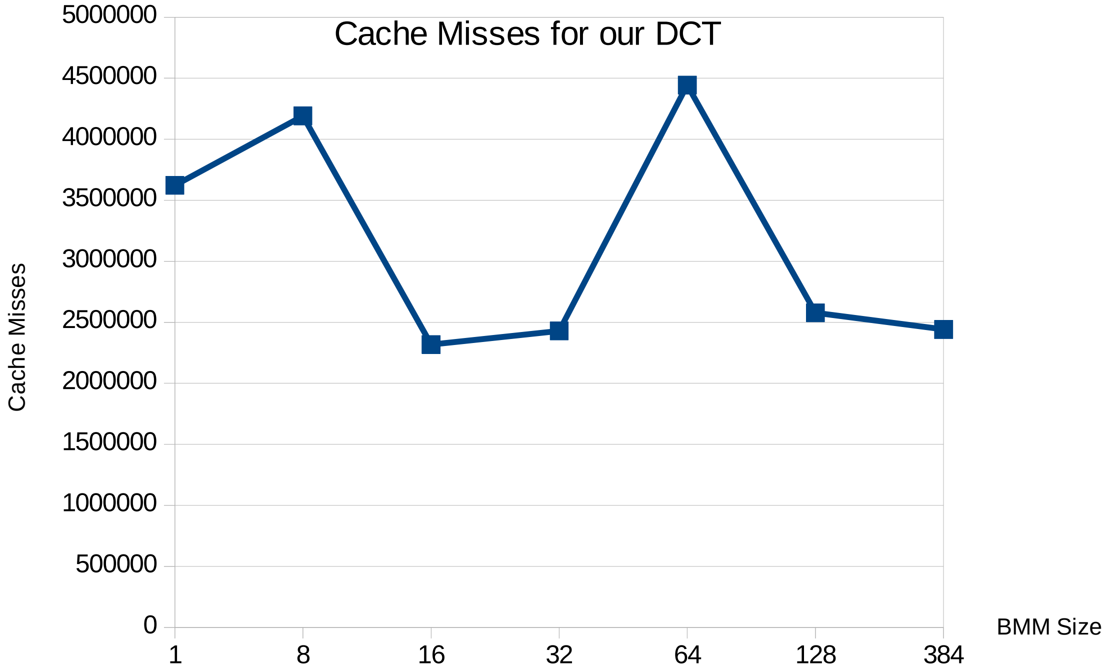
Patrick Hanrahan PID: A53204304
Benjamin Hobbs 11317149

WES237B

Assignment 3

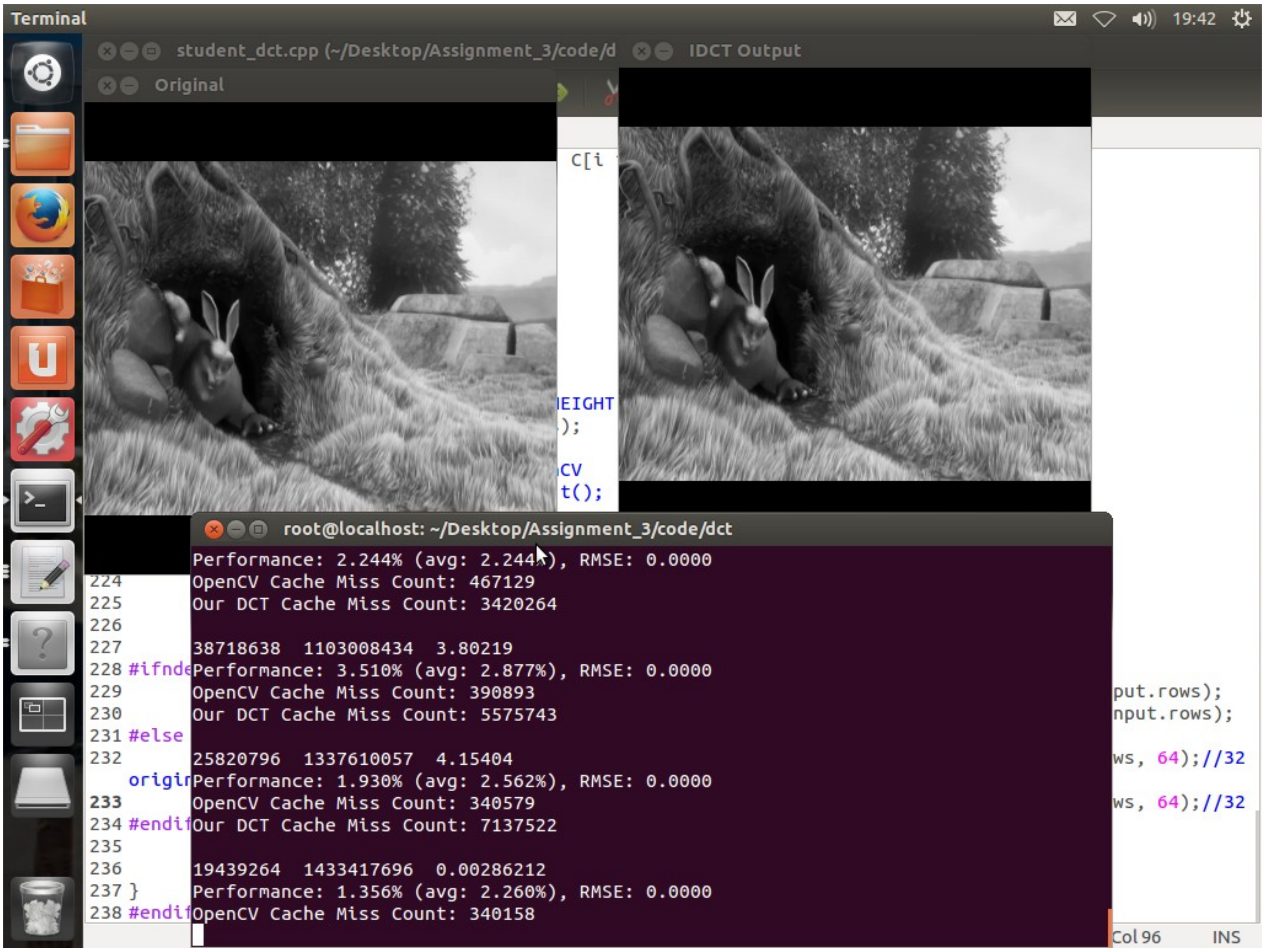
Part 1. DCT Cache misses





The most noticeable observation of the cache miss count is that the OpenCV command implementation was much more efficient with roughly 10 times less cache misses. This means that the cache memory is being accessed more successfully and with a greater “hits” percentage than our own method (without the OpenCV commands).

It is also observed that for the OpenCV implementation, the optimal Block sizes for BMM were 16 and 64. For the non-OpenCV implementation, 16, 32, 128, and 384 were optimal.

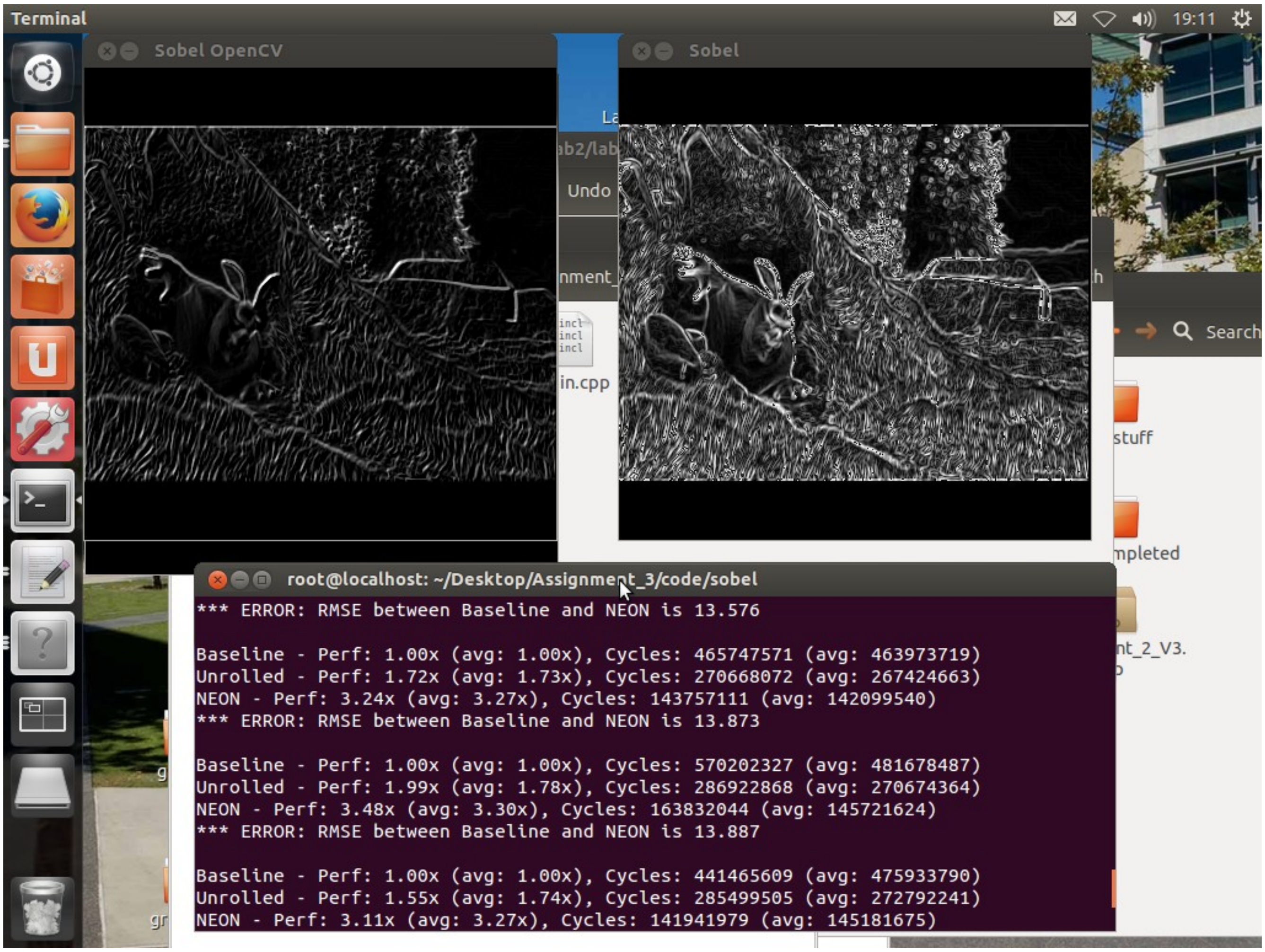

Terminal
19:42
student_dct.cpp (~/Desktop/Assignment_3/code/d
IDCT Output
Original
root@localhost: ~/Desktop/Assignment_3/code/dct
Col 96
INS

```
Performance: 2.244% (avg: 2.244%), RMSE: 0.0000
OpenCV Cache Miss Count: 467129
Our DCT Cache Miss Count: 3420264

38718638  1103008434  3.80219
Performance: 3.510% (avg: 2.877%), RMSE: 0.0000
OpenCV Cache Miss Count: 390893
Our DCT Cache Miss Count: 5575743

25820796  1337610057  4.15404
Performance: 1.930% (avg: 2.562%), RMSE: 0.0000
OpenCV Cache Miss Count: 340579
Our DCT Cache Miss Count: 7137522

19439264  1433417696  0.00286212
Performance: 1.356% (avg: 2.260%), RMSE: 0.0000
OpenCV Cache Miss Count: 340158
```

Part 2:

For Cache unrolling we reduced cache size by taking out a lot of the work from the for-loop. Instead of calling a short for-loop to do simple matrix multiplication and addition, we used 1 less for-loop and just did this step manually. Since accessing the for-loop takes a lot of time to check boundaries, but addition and multiplication is fast, this improves overall performance.

Part 3:



```
*** ERROR: RMSE between Baseline and NEON is 13.576

Baseline - Perf: 1.00x (avg: 1.00x), Cycles: 465747571 (avg: 463973719)
Unrolled - Perf: 1.72x (avg: 1.73x), Cycles: 270668072 (avg: 267424663)
NEON - Perf: 3.24x (avg: 3.27x), Cycles: 143757111 (avg: 142099540)
*** ERROR: RMSE between Baseline and NEON is 13.873

Baseline - Perf: 1.00x (avg: 1.00x), Cycles: 570202327 (avg: 481678487)
Unrolled - Perf: 1.99x (avg: 1.78x), Cycles: 286922868 (avg: 270674364)
NEON - Perf: 3.48x (avg: 3.30x), Cycles: 163832044 (avg: 145721624)
*** ERROR: RMSE between Baseline and NEON is 13.887

Baseline - Perf: 1.00x (avg: 1.00x), Cycles: 441465609 (avg: 475933790)
Unrolled - Perf: 1.55x (avg: 1.74x), Cycles: 285499505 (avg: 272792241)
NEON - Perf: 3.11x (avg: 3.27x), Cycles: 141941979 (avg: 145181675)
```

With our Neon implementation of the Sobel Filter we were able to complete the image process with approximately 3 times less cycles (145,721,624 vs 481,678,487 average cycles1), as seen in the image above. This proves that using the ARM NEON instructions to store the image data is at least 3 times as efficient as the baseline rolled-up method.

We reduced the number of for loops used, following the program architecture of the “unrolled” method, but this time used the NEON commands for the 2D convolution using the Sobel filter kernel.