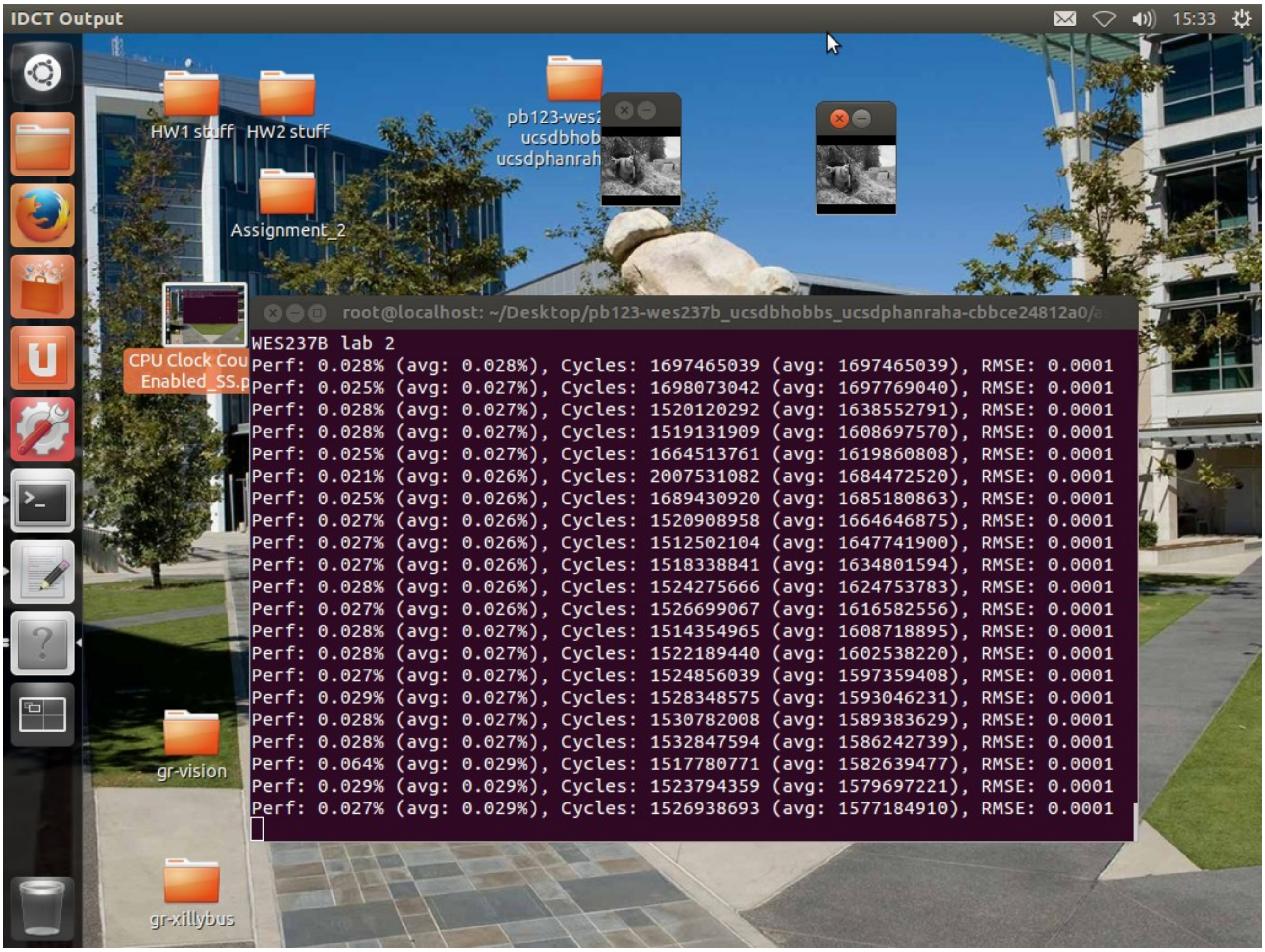
Patrick Hanrahan A53204304
Benjamin Hobbs
WES237b
Assignment 2 Report

Part 1: Look Up Table (LUT)

1. Performance and Cycle Count for an input size of 64 without optimizations:
   Performance Average: 0.029%
   Cycle Avg: 1,577,184,910

Figure 1: Naive DCT approach without optimizations, with input image Size 64



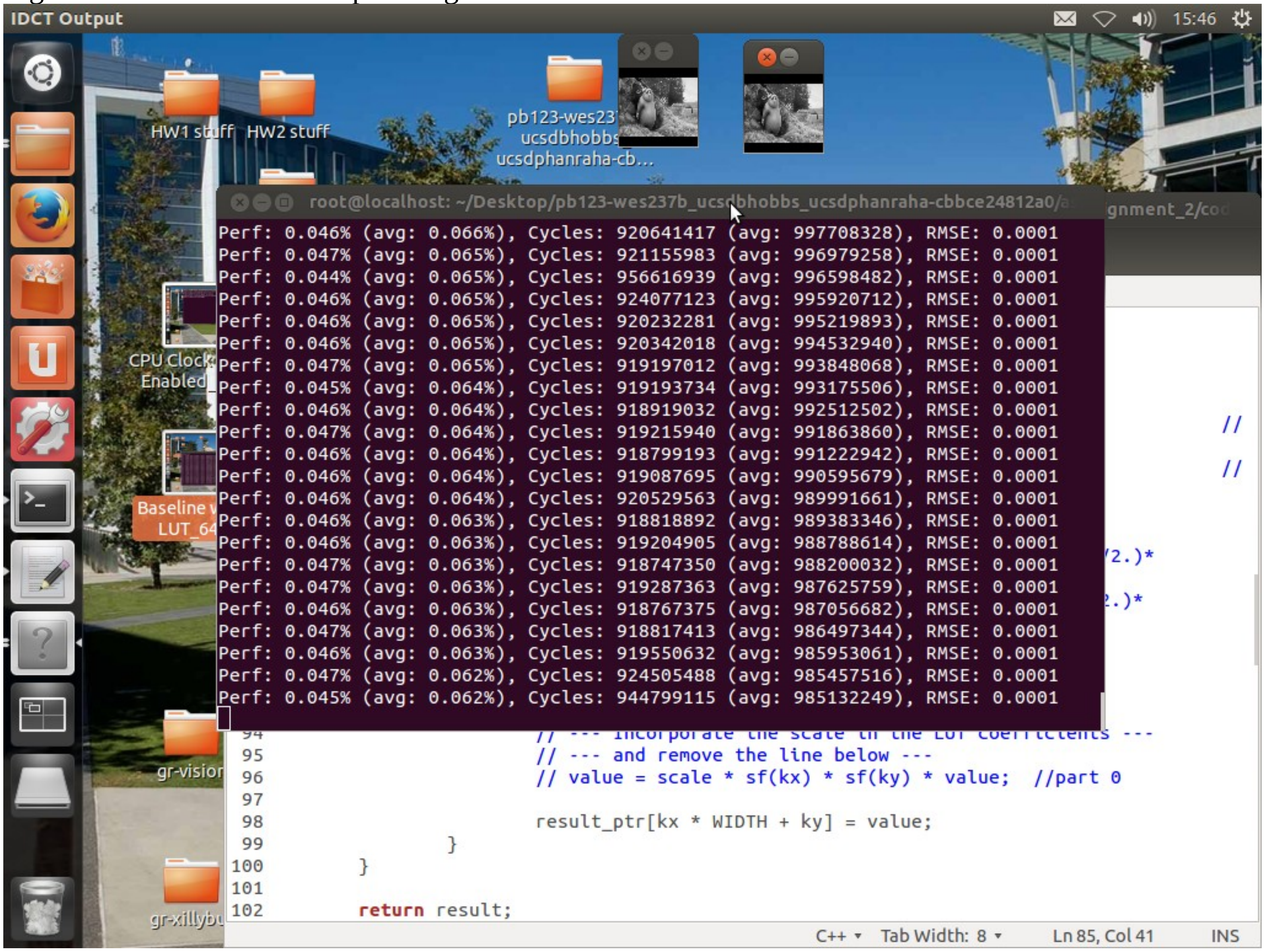
```
WES237B lab 2
Perf: 0.028% (avg: 0.028%), Cycles: 1697465039 (avg: 1697465039), RMSE: 0.0001
Perf: 0.025% (avg: 0.027%), Cycles: 1698073042 (avg: 1697769040), RMSE: 0.0001
Perf: 0.028% (avg: 0.027%), Cycles: 1520120292 (avg: 1638552791), RMSE: 0.0001
Perf: 0.028% (avg: 0.027%), Cycles: 1519131909 (avg: 1608697570), RMSE: 0.0001
Perf: 0.025% (avg: 0.027%), Cycles: 1664513761 (avg: 1619860808), RMSE: 0.0001
Perf: 0.021% (avg: 0.026%), Cycles: 2007531082 (avg: 1684472520), RMSE: 0.0001
Perf: 0.025% (avg: 0.026%), Cycles: 1689430920 (avg: 1685180863), RMSE: 0.0001
Perf: 0.027% (avg: 0.026%), Cycles: 1520908958 (avg: 1664646875), RMSE: 0.0001
Perf: 0.027% (avg: 0.026%), Cycles: 1512502104 (avg: 1647741900), RMSE: 0.0001
Perf: 0.027% (avg: 0.026%), Cycles: 1518338841 (avg: 1634801594), RMSE: 0.0001
Perf: 0.028% (avg: 0.026%), Cycles: 1524275666 (avg: 1624753783), RMSE: 0.0001
Perf: 0.027% (avg: 0.026%), Cycles: 1526699067 (avg: 1616582556), RMSE: 0.0001
Perf: 0.028% (avg: 0.027%), Cycles: 1514354965 (avg: 1608718895), RMSE: 0.0001
Perf: 0.028% (avg: 0.027%), Cycles: 1522189440 (avg: 1602538220), RMSE: 0.0001
Perf: 0.027% (avg: 0.027%), Cycles: 1524856039 (avg: 1597359408), RMSE: 0.0001
Perf: 0.029% (avg: 0.027%), Cycles: 1528348575 (avg: 1593046231), RMSE: 0.0001
Perf: 0.028% (avg: 0.027%), Cycles: 1530782008 (avg: 1589383629), RMSE: 0.0001
Perf: 0.028% (avg: 0.027%), Cycles: 1532847594 (avg: 1586242739), RMSE: 0.0001
Perf: 0.064% (avg: 0.029%), Cycles: 1517780771 (avg: 1582639477), RMSE: 0.0001
Perf: 0.029% (avg: 0.029%), Cycles: 1523794359 (avg: 1579697221), RMSE: 0.0001
Perf: 0.027% (avg: 0.029%), Cycles: 1526938693 (avg: 1577184910), RMSE: 0.0001
```

2-4. Complete LUT code with scaling may be found in our repository in the assignment_2/code/src directory.

5. Performance and Cycle Count for an input size of 64 with LUT**:
   Performance Average: 0.060%
   Cycles Avg: 978,108,351

***Note: Please uncomment the “Part 1 Naive LUT implementation” to test.*

Figure 2: Naive LUT with input image Size 64



```
Perf: 0.046% (avg: 0.066%), Cycles: 920641417 (avg: 997708328), RMSE: 0.0001
Perf: 0.047% (avg: 0.065%), Cycles: 921155983 (avg: 996979258), RMSE: 0.0001
Perf: 0.044% (avg: 0.065%), Cycles: 956616939 (avg: 996598482), RMSE: 0.0001
Perf: 0.046% (avg: 0.065%), Cycles: 924077123 (avg: 995920712), RMSE: 0.0001
Perf: 0.046% (avg: 0.065%), Cycles: 920232281 (avg: 995219893), RMSE: 0.0001
Perf: 0.046% (avg: 0.065%), Cycles: 920342018 (avg: 994532940), RMSE: 0.0001
Perf: 0.047% (avg: 0.065%), Cycles: 919197012 (avg: 993848068), RMSE: 0.0001
Perf: 0.045% (avg: 0.064%), Cycles: 919193734 (avg: 993175506), RMSE: 0.0001
Perf: 0.046% (avg: 0.064%), Cycles: 918919032 (avg: 992512502), RMSE: 0.0001
Perf: 0.047% (avg: 0.064%), Cycles: 919215940 (avg: 991863860), RMSE: 0.0001
Perf: 0.046% (avg: 0.064%), Cycles: 918799193 (avg: 991222942), RMSE: 0.0001
Perf: 0.046% (avg: 0.064%), Cycles: 919087695 (avg: 990595679), RMSE: 0.0001
Perf: 0.046% (avg: 0.064%), Cycles: 920529563 (avg: 989991661), RMSE: 0.0001
Perf: 0.046% (avg: 0.063%), Cycles: 918818892 (avg: 989383346), RMSE: 0.0001
Perf: 0.046% (avg: 0.063%), Cycles: 919204905 (avg: 988788614), RMSE: 0.0001
Perf: 0.047% (avg: 0.063%), Cycles: 918747350 (avg: 988200032), RMSE: 0.0001
Perf: 0.047% (avg: 0.063%), Cycles: 919287363 (avg: 987625759), RMSE: 0.0001
Perf: 0.046% (avg: 0.063%), Cycles: 918767375 (avg: 987056682), RMSE: 0.0001
Perf: 0.047% (avg: 0.063%), Cycles: 918817413 (avg: 986497344), RMSE: 0.0001
Perf: 0.046% (avg: 0.063%), Cycles: 919550632 (avg: 985953061), RMSE: 0.0001
Perf: 0.047% (avg: 0.062%), Cycles: 924505488 (avg: 985457516), RMSE: 0.0001
Perf: 0.045% (avg: 0.062%), Cycles: 944799115 (avg: 985132249), RMSE: 0.0001
```

```
                        // --- incorporate the scale in the LUT coefficients ---
                        // --- and remove the line below ---
                        // value = scale * sf(kx) * sf(ky) * value;  //part 0

                        result_ptr[kx * WIDTH + ky] = value;
                }
        }

        return result;
```

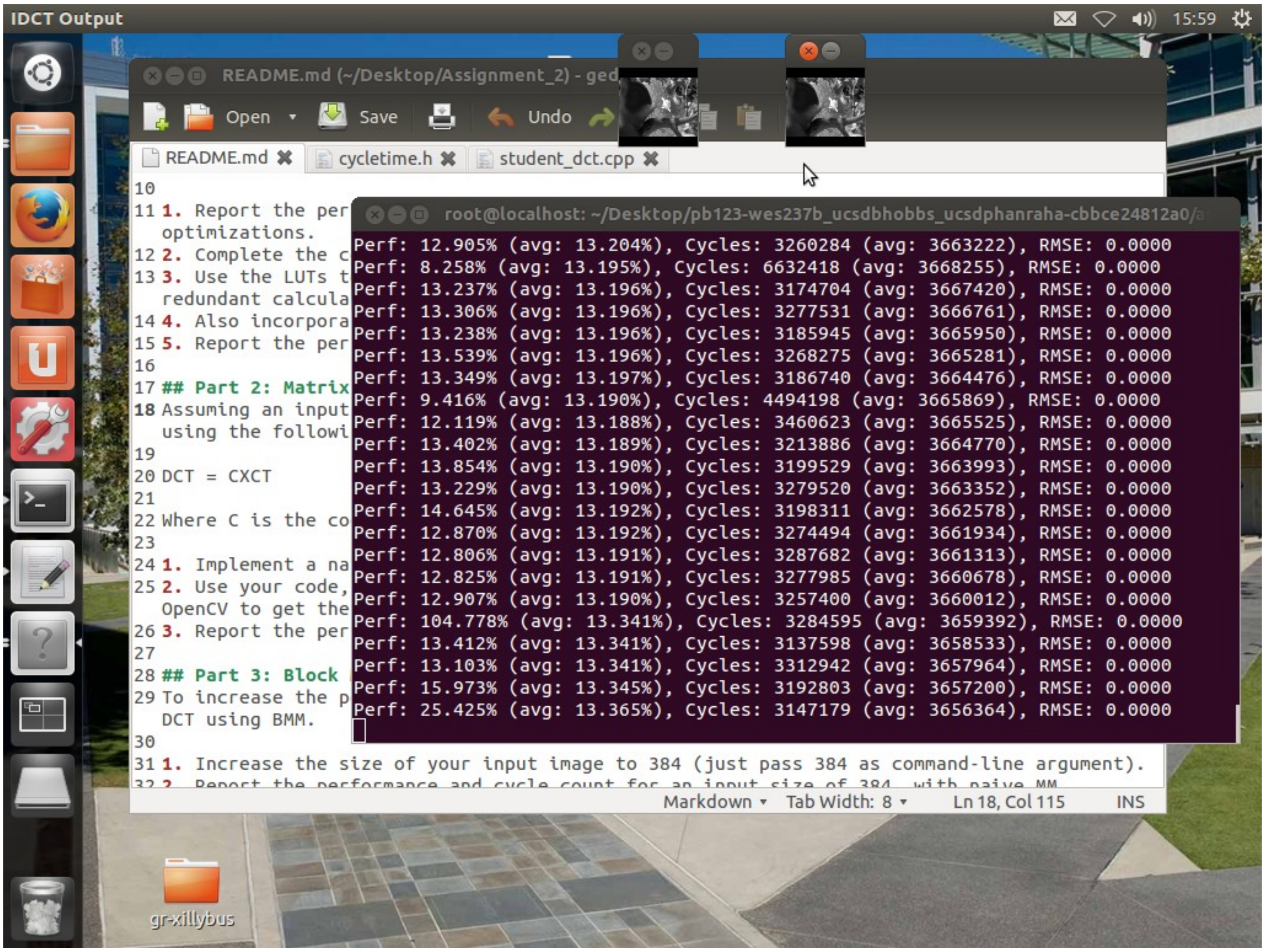
Part 2: Matrix Multiplication (MM)

1-2. Implemented naive matrix multiplication code may be found in our repo in the assignment_2/code/src directory.

3. Performance and Cycle Count for an input size of 64 with the naive Matrix Multiplication implemented**:
    Performance avg: 13.303%
    Cycles Avg: 3,706,906

***Note: Please uncomment the "Part 2: DCT as matrix multiplication" to test.*

Figure 3: Matrix Multiplication with input image Size 64



```
1. Report the per
optimizations.
2. Complete the c
3. Use the LUTs t
redundant calcula
4. Also incorpora
5. Report the per

## Part 2: Matrix
Assuming an input
using the followi

DCT = CXCT

Where C is the co

1. Implement a na
2. Use your code,
OpenCV to get the
3. Report the per

## Part 3: Block
To increase the p
DCT using BMM.

1. Increase the size of your input image to 384 (just pass 384 as command-line argument).
```

```
Perf: 12.905% (avg: 13.204%), Cycles: 3260284 (avg: 3663222), RMSE: 0.0000
Perf: 8.258% (avg: 13.195%), Cycles: 6632418 (avg: 3668255), RMSE: 0.0000
Perf: 13.237% (avg: 13.196%), Cycles: 3174704 (avg: 3667420), RMSE: 0.0000
Perf: 13.306% (avg: 13.196%), Cycles: 3277531 (avg: 3666761), RMSE: 0.0000
Perf: 13.238% (avg: 13.196%), Cycles: 3185945 (avg: 3665950), RMSE: 0.0000
Perf: 13.539% (avg: 13.196%), Cycles: 3268275 (avg: 3665281), RMSE: 0.0000
Perf: 13.349% (avg: 13.197%), Cycles: 3186740 (avg: 3664476), RMSE: 0.0000
Perf: 9.416% (avg: 13.190%), Cycles: 4494198 (avg: 3665869), RMSE: 0.0000
Perf: 12.119% (avg: 13.188%), Cycles: 3460623 (avg: 3665525), RMSE: 0.0000
Perf: 13.402% (avg: 13.189%), Cycles: 3213886 (avg: 3664770), RMSE: 0.0000
Perf: 13.854% (avg: 13.190%), Cycles: 3199529 (avg: 3663993), RMSE: 0.0000
Perf: 13.229% (avg: 13.190%), Cycles: 3279520 (avg: 3663352), RMSE: 0.0000
Perf: 14.645% (avg: 13.192%), Cycles: 3198311 (avg: 3662578), RMSE: 0.0000
Perf: 12.870% (avg: 13.192%), Cycles: 3274494 (avg: 3661934), RMSE: 0.0000
Perf: 12.806% (avg: 13.191%), Cycles: 3287682 (avg: 3661313), RMSE: 0.0000
Perf: 12.825% (avg: 13.191%), Cycles: 3277985 (avg: 3660678), RMSE: 0.0000
Perf: 12.907% (avg: 13.190%), Cycles: 3257400 (avg: 3660012), RMSE: 0.0000
Perf: 104.778% (avg: 13.341%), Cycles: 3284595 (avg: 3659392), RMSE: 0.0000
Perf: 13.412% (avg: 13.341%), Cycles: 3137598 (avg: 3658533), RMSE: 0.0000
Perf: 13.103% (avg: 13.341%), Cycles: 3312942 (avg: 3657964), RMSE: 0.0000
Perf: 15.973% (avg: 13.345%), Cycles: 3192803 (avg: 3657200), RMSE: 0.0000
Perf: 25.425% (avg: 13.365%), Cycles: 3147179 (avg: 3656364), RMSE: 0.0000
```

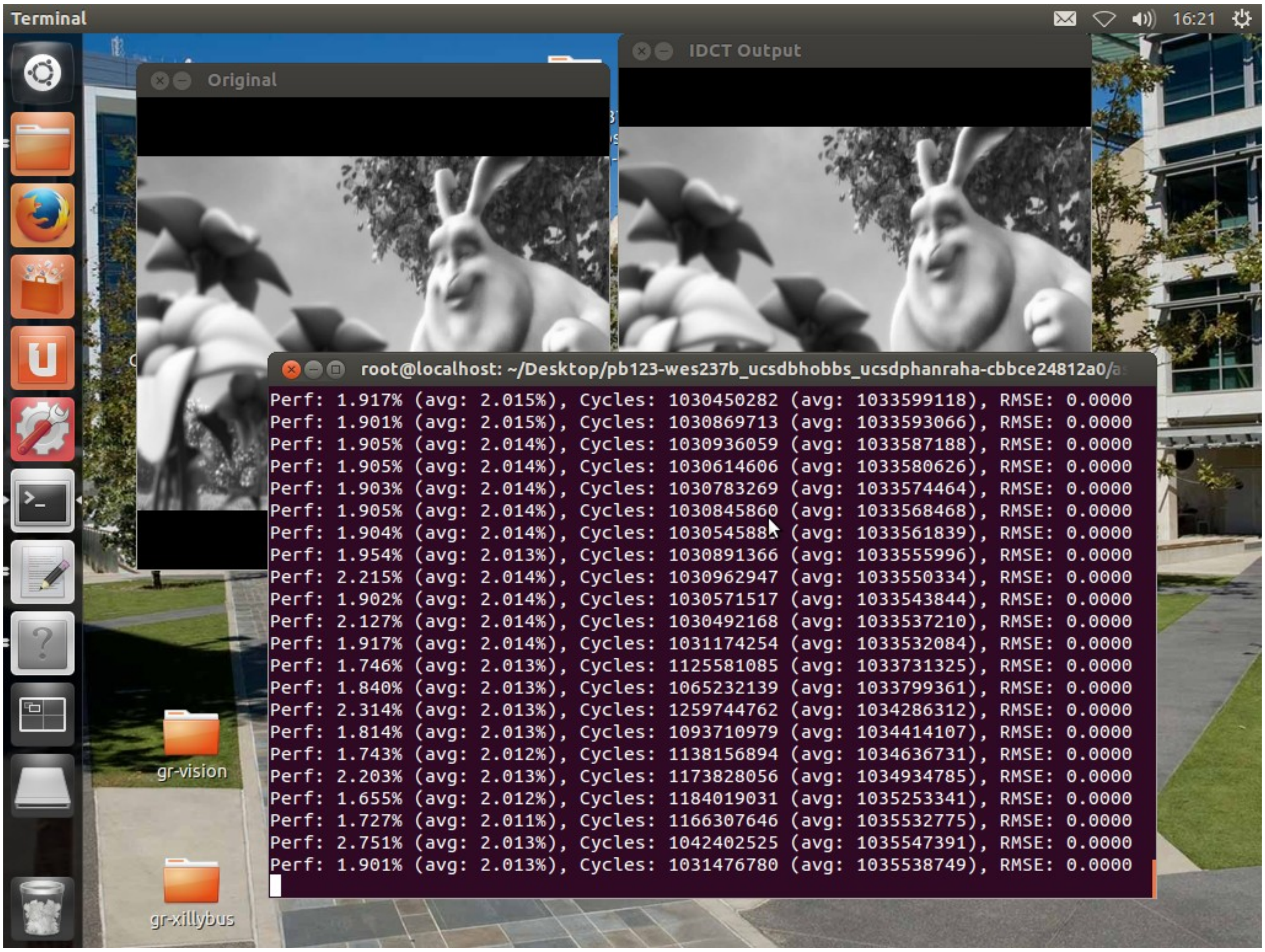
Part 3: Block Matrix Multiplication (BMM)

1-2. Performance and Cycle Count for an input size of 384 with the naive Matrix Multiplication implemented**:

Performance avg: 2.020%
Cycles Avg: 1,037,163,938
RMSE: 0.000

****Note: *Please uncomment the "Part 2: DCT as matrix multiplication" to test.*

Figure 3: Matrix Multiplication with input image Size 384



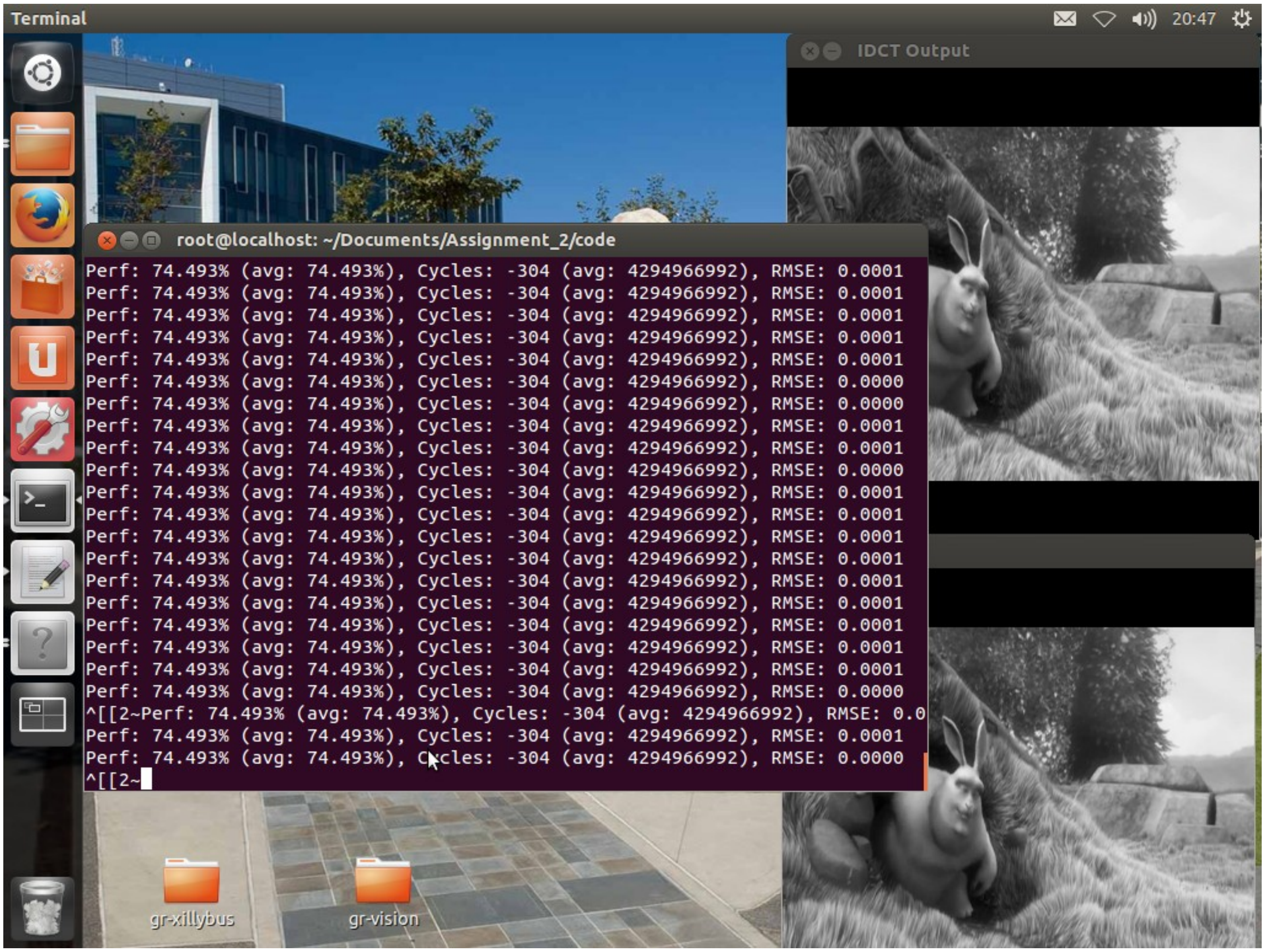
```
Perf: 1.917% (avg: 2.015%), Cycles: 1030450282 (avg: 1033599118), RMSE: 0.0000
Perf: 1.901% (avg: 2.015%), Cycles: 1030869713 (avg: 1033593066), RMSE: 0.0000
Perf: 1.905% (avg: 2.014%), Cycles: 1030936059 (avg: 1033587188), RMSE: 0.0000
Perf: 1.905% (avg: 2.014%), Cycles: 1030614606 (avg: 1033580626), RMSE: 0.0000
Perf: 1.903% (avg: 2.014%), Cycles: 1030783269 (avg: 1033574464), RMSE: 0.0000
Perf: 1.905% (avg: 2.014%), Cycles: 1030845860 (avg: 1033568468), RMSE: 0.0000
Perf: 1.904% (avg: 2.014%), Cycles: 103054588[illegible] (avg: 1033561839), RMSE: 0.0000
Perf: 1.954% (avg: 2.013%), Cycles: 1030891366 (avg: 1033555996), RMSE: 0.0000
Perf: 2.215% (avg: 2.014%), Cycles: 1030962947 (avg: 1033550334), RMSE: 0.0000
Perf: 1.902% (avg: 2.014%), Cycles: 1030571517 (avg: 1033543844), RMSE: 0.0000
Perf: 2.127% (avg: 2.014%), Cycles: 1030492168 (avg: 1033537210), RMSE: 0.0000
Perf: 1.917% (avg: 2.014%), Cycles: 1031174254 (avg: 1033532084), RMSE: 0.0000
Perf: 1.746% (avg: 2.013%), Cycles: 1125581085 (avg: 1033731325), RMSE: 0.0000
Perf: 1.840% (avg: 2.013%), Cycles: 1065232139 (avg: 1033799361), RMSE: 0.0000
Perf: 2.314% (avg: 2.013%), Cycles: 1259744762 (avg: 1034286312), RMSE: 0.0000
Perf: 1.814% (avg: 2.013%), Cycles: 1093710979 (avg: 1034414107), RMSE: 0.0000
Perf: 1.743% (avg: 2.012%), Cycles: 1138156894 (avg: 1034636731), RMSE: 0.0000
Perf: 2.203% (avg: 2.013%), Cycles: 1173828056 (avg: 1034934785), RMSE: 0.0000
Perf: 1.655% (avg: 2.012%), Cycles: 1184019031 (avg: 1035253341), RMSE: 0.0000
Perf: 1.727% (avg: 2.011%), Cycles: 1166307646 (avg: 1035532775), RMSE: 0.0000
Perf: 2.751% (avg: 2.013%), Cycles: 1042402525 (avg: 1035547391), RMSE: 0.0000
Perf: 1.901% (avg: 2.013%), Cycles: 1031476780 (avg: 1035538749), RMSE: 0.0000
```

3. Implementation of the Block Multiplication Matrix code may be found in our repo in the assignment_2/code/src directory**.

***Note: Please uncomment the "BMM Part 3" to test.*

4. The optimal block matrix size was 16. This was found by trial and error.

5. We ran into difficulty getting the performance readings working for the BMM. However, we were able to get the RMSE down to 0.



```
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0000
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0000
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0000
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0000
^[[2~Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0001
Perf: 74.493% (avg: 74.493%), Cycles: -304 (avg: 4294966992), RMSE: 0.0000
^[[2~
```