Kayenat-Patil-HW-1

**Question 1:**

To do:

1. Plot Histograms for tex.din and spice.din
2. What is the frequency of writes? What is the frequency of reads? Please comment on these results.

Done:

Since the address were given in Hexadecimal, firstly all the addresses were converted to Decimal.

For doing this, I used Ubuntu 18.04 LST and ran the command ‘awk posix ‘{printf(“%d \t %d \n”, $1, “0x” $2)}’ fname > fnameOut’

Where fname is tex and spice and fnameOut is the file I used for storing the result so that the input remained unchanged and can be referenced to at any point if needed. Furthermore, sorting can be done so that the range of address are sequential.

I ran the following command for sorting ‘Sort -k 2n fname > fnameOut’

For plotting the histograms, I used Python as data visualization is much easier, it can be done using gnuplots as well. The number of bins I used was 10, or the formula can be used is bin = (max-min)/width to get the value for bin. Since I used google colabs to get the plots, it couldn’t handle the bin range, so I put it as 10 instead.

I converted the text files to csv files as data frames can be handles easier than text files. I ran the following code:

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

df = pd.read\_csv('x.csv') #x is the csv file, tex and spice

add = df["Address"]

plt.hist(add, bins =10, edgecolor = "black" )# number of bins can be

# adjusted according to the

# data

The resulting histograms are as follows:

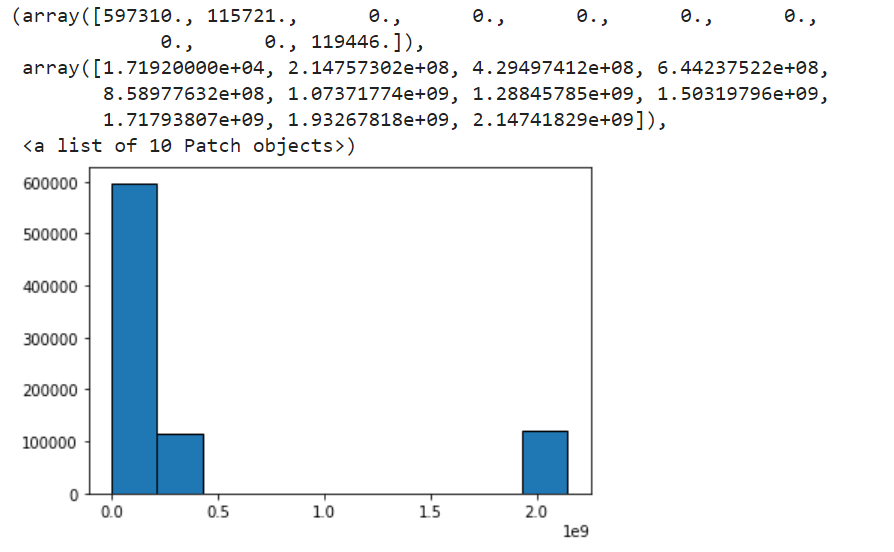


Fig 1: Histogram for tex

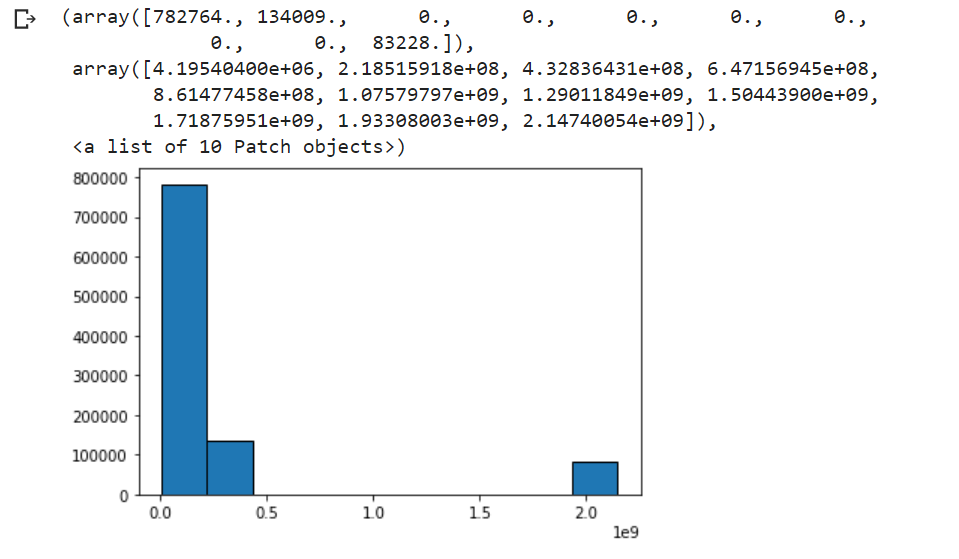


Fig2: Histogram for spice

The Frequency of read for tex.din is 130655 and writes is 104513

The Frequency of read for spice.din is 150699 and writes is 66538

Inference:

It can be concluded that for both tex and spice the number of read is more than writes. This could be because write is an expense operation (when compared to read) as it is expensive change/append a record. Also, read is a faster instruction than write.

**Question 2:**

I used python for matrix multiplication. The code is as follows:

import numpy as np

import time

start = time.time()

Mat1 = np.random.randint(10, size=(10, 20))

Mat2 = np.random.randint(10, size=(20, 35))

Mat3 = np.random.uniform(10, size=(1000, 2000))

Mat4 = np.random.uniform(10, size=(2000, 3500))

Result = [[0 for x in range(2000)] for y in range(3000)]

# explicit for loops

for i in range(len(Mat1)):

    for j in range(len(Mat2[0])):

        for k in range(len(Mat2)):

            Result[i][j] += Mat1[i][k] \* Mat2[k][j]

end = time.time()

print(Result“\n”)

print(end - start)

Note, here Mat1 an Mat2 generate random values for int matrices and Mat3 and Mat4 generate random values for double matrices. To get the output for double simply switch the for loops and put Mat3 and Mat4 instead of Mat1 and Mat2, respectively.

As Follows:

import numpy as np

import time

start = time.time()

Mat1 = np.random.randint(10, size=(1000, 2000))

Mat2 = np.random.randint(10, size=(2000, 3500))

Mat3 = np.random.uniform(10, size=(1000, 2000))

Mat4 = np.random.uniform(10, size=(2000, 3500))

Result = [[0 for x in range(2000)] for y in range(3000)]

# explicit for loops

for i in range(len(Mat3)):

    for j in range(len(Mat4[0])):

        for k in range(len(Mat4)):

            Result[i][j] += Mat3[i][k] \* Mat4[k][j]

end = time.time()

print(Result“\n”)

print(end - start)

The following table shows time taken for the program to run

|  |  |  |  |
| --- | --- | --- | --- |
| Int matrices on Windows | Double matrices on Windows | Int matrices on Linux | Double matrices on Linux |
| 19.385268211364746 | 0.7103395462036133 | 16.79010558128357 | 0.32285547256469727 |
| 19.156340837478638 | 0.6925816535949707 | 17.15945029258728 | 0.32383155822753906 |
| 19.540745735168457 | 0.6440162658691406 | 16.746391773223877 | 0.3336496353149414 |
| 19.158682107925415 | 0.6615645885467529 | 16.646567821502686 | 0.32715654373168945 |
| 19.062840938568115 | 0.6550252437591553 | 16.55021381378174 | 0.32709503173828125 |
| 19.390235424041748 | 0.6826381683349609 | 16.4542396068573 | 0.3290591239929199 |
| 19.209683418273926 | 0.6526508331298828 | 16.928001403808594 | 0.32663917541503906 |

Note: The time is based on the number of rows and columns mentioned in the code above.

Averages:

1. Int matrices on Windows: 19.26
2. Double matrices on Windows: 0.670
3. Int matrices on Linux: 16.74
4. Double matrices on Linux: 0.326

Inference:

By the above table its observed that Linux system is much faster for all the times the same piece of code was run on the two systems. If we consider the case for double, the Linux system was almost twice as fast.

Is the performance ratio the same as the clock rate ratio of the two systems?

The performance ratio of the two system comes out to be as 0.149 if we take the average time take to run the code. Which indicates that Windows was slower as it was taken as the numerator, which confirms our findings.

The clock rate ratio is 1.24 approximately, so no we cannot say that the performance ration is the same as clock rate ratio of the two systems.

Which is more cost effective?

If we strictly look at Linux VS Windows in terms of the calculations that the assignment demanded then the answer is easy. Linux is cost effect as it almost costs nothing to run the code on Linux and needs no licensing fee at all (like using software like Ubuntu, Fedora, etc.) It is open source and anybody with a computer can get it free of cost. Where as Widows is a paid operating system. Although there are paid Linux platforms like Red Hat, but for the purposes of this project, since I ran the code on Ubuntu, it cost me nothing to run it on the server.

Description of systems:

1. Windows:
2. Manufacturer: Dell
3. Make/Model: Inspiron 14
4. CPU type: AMD Ryzen 7 2.48GHz
5. Memory: 16GB Memory - 512GB SSD
6. Language: Python on Google Colabs
7. Linux:
8. Manufacturer: Ubuntu
9. Make/Model: 18.04 LTS
10. CPU type: intel i7 2GHz
11. Memory: 24GB
12. Language: Python via Terminal