Jessica Fontaine
CS 402

Homework 1

1a. “Using trace files, i.e. files that contain addresses issued by some CPU to execute some application(s), draw the histogram of address distribution for each of them.”

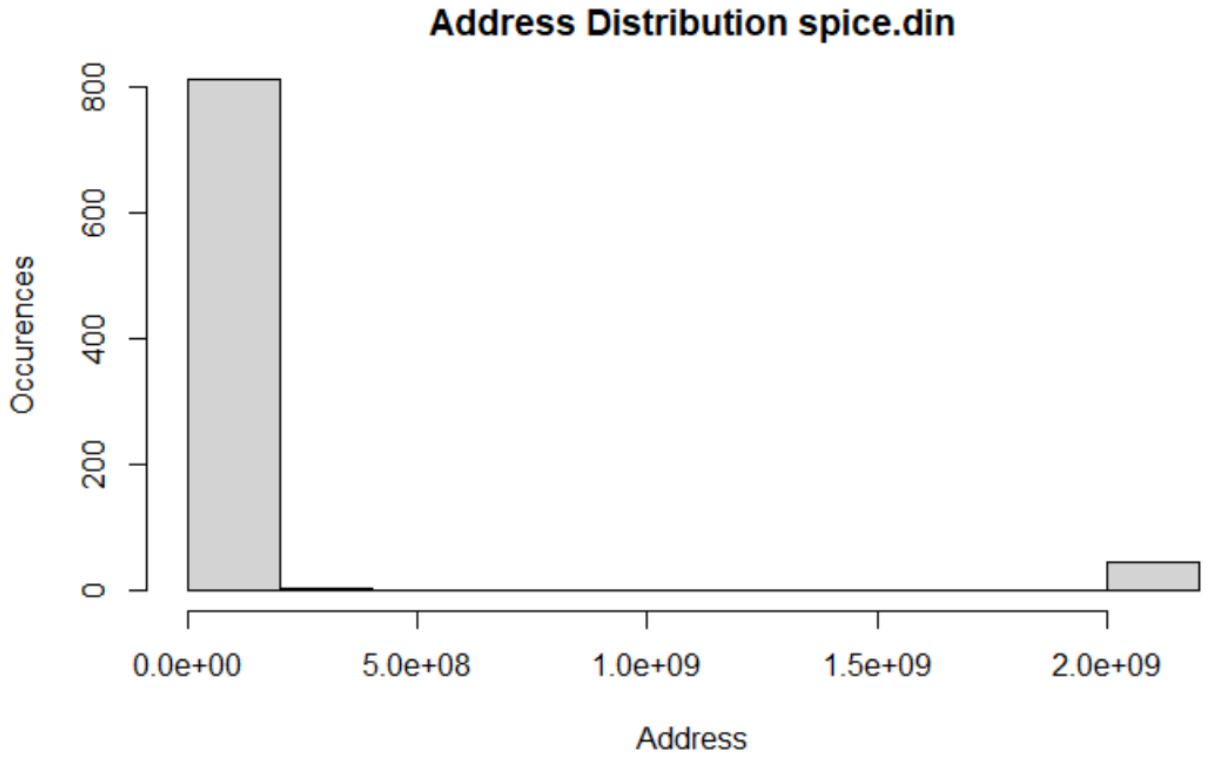

Fig. 1: Address distribution of ~800 addresses sampled from the spice.din data source.

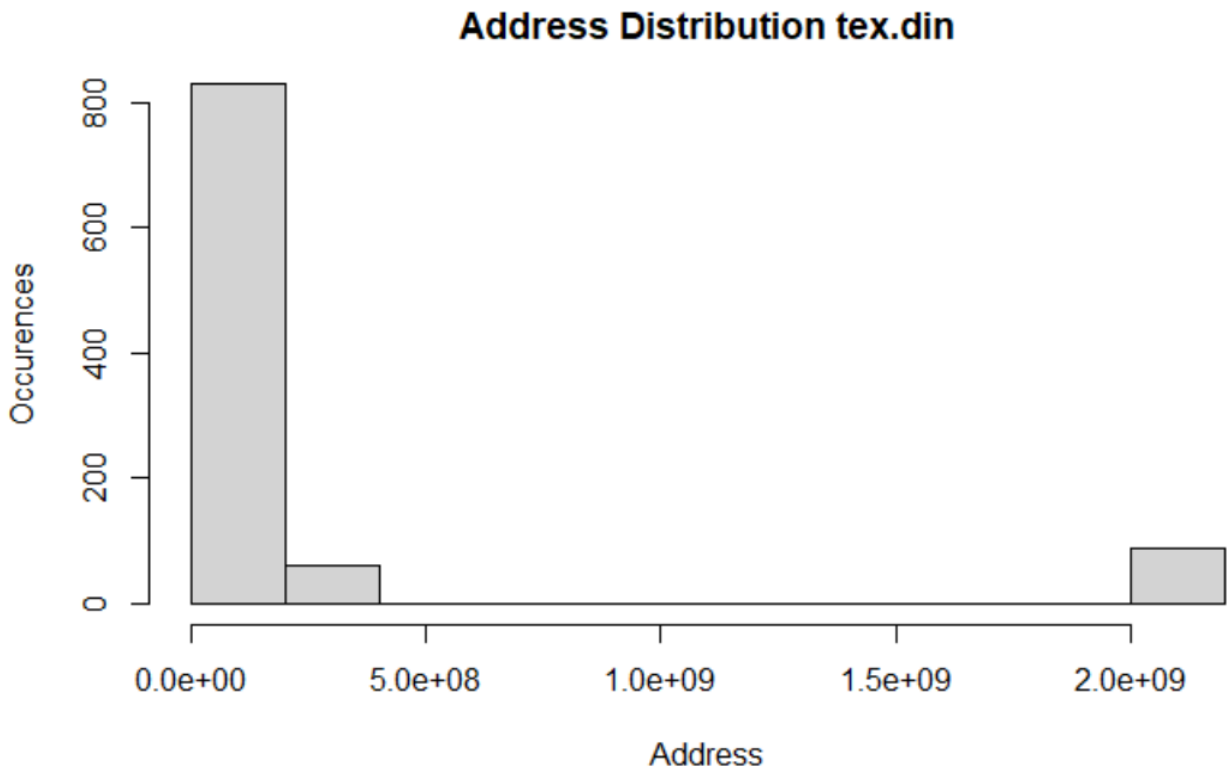

Fig. 2: Address distribution of ~800 addresses sampled from the tex.din data source.

“Comment based on the histograms (5).”

It can be observed that both trace files have a similar distribution of memory addresses.This is surprising, given the diversity of programs that were used to generate the data sets: Fig.1 contains the trace obtained by executing a general-purpose, open-source analog electronic circuit simulator, while Fig. 2 comes from a typesetting application. Both trace files also contain a large gap in memory addresses. I hypothesize that there is a cost benefit to grouping memory addresses together in such a way. It is also worth noting that tex.din appears to possess a slightly greater number of memory addresses where you have non-zero values on Oy. It is possible that this could be due to sampling error. Per the instructions for this assignment, I attempted to select a range of addresses with non-zero values on Oy. Further analysis using the complete data set would be needed to rule out sampling issues.

1b. “What is the frequency of writes (5)? What is the frequency of reads (5)? Please comment on these results (5).”

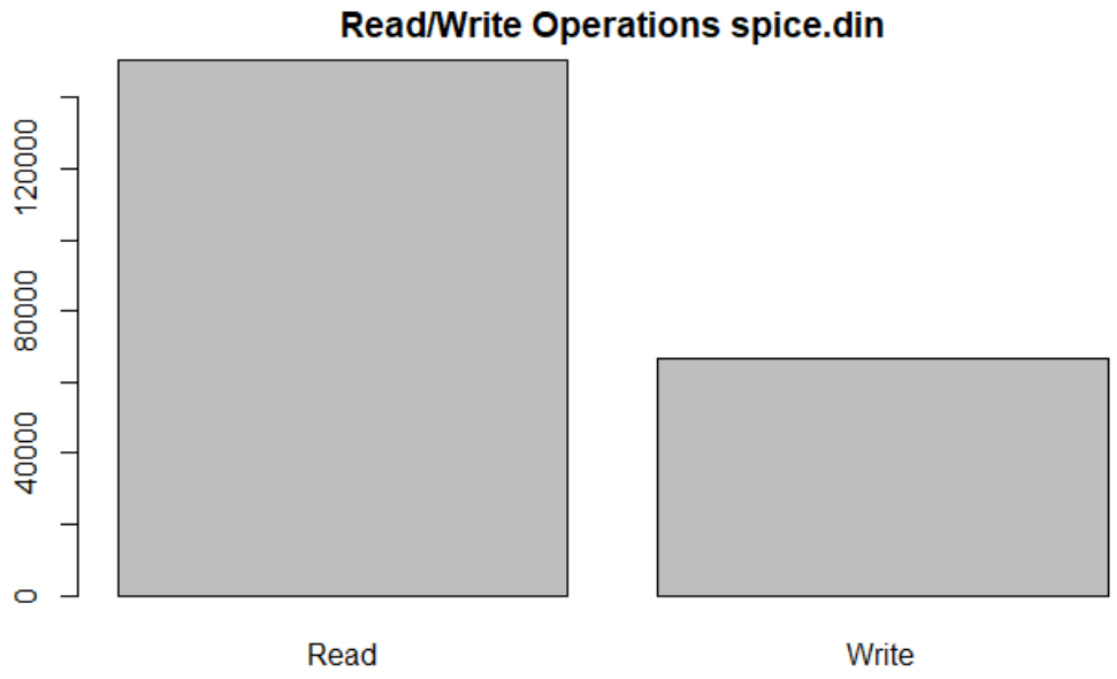

Fig. 3: Count of read/write operations from the spice.tin data source.

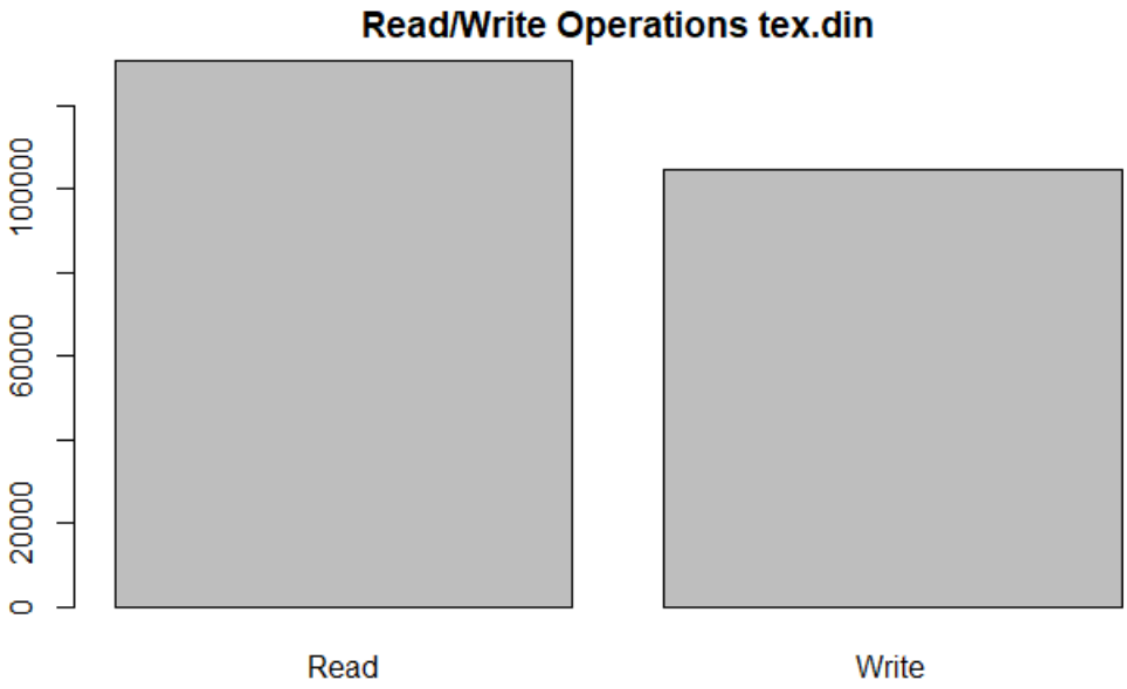

Fig. 3: Count of read/write operations from the tex.tin data source.

Spice.din contains 150699 reads and 66538 writes. Tex.din contains 130655 reads and 104513 writes. It can be observed that both trace files have a greater number of read operations than

write operations. This makes sense, given that it may take several read operations to move between data.

2a. “Write a program, using your favorite programming language, that multiplies two rectangular matrices -- please no square matrices -- whose elements are randomly generated. You will have two versions of the program, one in which matrix elements are integers and another one where they are real numbers (double)”

Please see attached 2a_matrix.py file.

Measure the time it takes each program to complete (2x5) and then compare the performance of the two systems (5).

System #1

Integer Matrix

Attempt	**Runtime (seconds)**
1	9.688995838165283
2	9.689968824386597
3	9.64103388786316
4	9.630005836486816
5	9.62099289894104
6	9.612997770309448
7	9.59100079536438
8	9.547028064727783
9	9.575000286102295
10	9.36299991607666
Average	9.596002412

Double Matrix

Attempt	**Runtime (seconds)**
1	9.787999629974365

2	9.79799771308899
3	9.810001611709595
4	10.014002323150635
5	10.047993183135986
6	10.032005548477173
7	10.032028675079346
8	10.098002433776855
9	10.129995346069336
10	10.015979051589966
Average	9.976600552

System #2

Integer Matrix

Attempt	**Runtime (seconds)**
1	15.41690707206726
2	15.447219848632812
3	15.512372970581055
4	16.171874046325684
5	15.590062856674194
6	15.003020763397217
7	15.074910402297974
8	15.032307386398315
9	15.385481834411621
10	15.078783988952637
Average	15.37129412

Double Matrix

Attempt	**Runtime (seconds)**

1	13.366889953613281
2	13.37619686126709
3	13.68432879447937
4	13.326965093612671
5	13.397411823272705
6	13.32072901725769
7	13.674546003341675
8	13.842729091644287
9	13.387145042419434
10	14.215564727783203
Average	13.55925064

Performance Comparison

Matrix Type	**Avg Runtime (seconds) System #1**	**Avg Runtime (seconds) System #2**
Integer	9.596002412	15.37129412
Double	9.976600552	13.55925064

System #1 was significantly faster than System #2 in both tests. It's worth noting that System #2 executed the double matrix multiplication faster than the integer matrix multiplication, but this was not the case for System #1. More investigation is needed to understand the relationship.

"Is the performance ratio the same as the clock rate ratio of the two systems (5)? "

No. Performance ratio and clock rate ratio are two different measurements. Performance ratio is defined as the ratio of measured output to expected output, while clock rate ratio is the speed ratio between the computer's front side bus and the CPU. The clock rate of System #1 is 2.5 GHz and System #2 is 2.8 GHz. Therefore, the clock rate ratio of the two systems is 2.5:2.8.

"Based on the retail price of the two systems, which one is more cost effective (5)?"

System #1 is a desktop computer manufactured by iBUYPOWER. The pre-tax retail price of this system is $1,349.99. System #2 is a 2019 13-inch MacBook Pro. This model is no longer new, but based on some initial research I found that the pre-tax retail price of this system is

approximately $1,299. This makes Systems #1 and #2 comparative in price. In terms of speed, System #1 outperformed System #2 by ~25% in the integer test and ~37% in the double test. Given the similarities in price, this makes System #1 more cost effective.

2b. “Change your multiplication algorithm and repeat the steps above.”

Please see attached 2b_matrix.py file for source code.

System #1

Integer Matrix

Attempt	**Runtime (seconds)**
1	0.014002323150634766
2	0.013037919998168945
3	0.013998270034790039
4	0.01400303840637207
5	0.013999462127685547
6	0.014001846313476562
7	0.013995647430419922
8	0.013978004455566406
9	0.014021873474121094
10	0.014997482299804688
Average	0.01400358677

Double Matrix

Attempt	**Runtime (seconds)**
1	0.0010044574737548828
2	0.0009620189666748047
3	0.0010013580322265625
4	0.0010001659393310547
5	0.0010037422180175781

6	0.0
7	0.0
8	0.0010025501251220703
9	0.0010020732879638672
10	0.0009968280792236328
Average	0.0007973194122

System #2

Integer Matrix

Attempt	**Runtime (seconds)**
1	0.021245956420898438
2	0.0212249755859375
3	0.02131199836730957
4	0.021132946014404297
5	0.021463871002197266
6	0.021265029907226562
7	0.020721912384033203
8	0.020924806594848633
9	0.021070003509521484
10	0.021081209182739258
Average	0.0211442709

Double Matrix

Attempt	**Runtime (seconds)**
1	0.0009889602661132812
2	0.0006327629089355469
3	0.00046515464782714844
4	0.0004930496215820312

5	0.00040984153747558594
6	0.00040602684020996094
7	0.00045013427734375
8	0.00043392181396484375
9	0.00040057884216308594
10	0.00040292739868164 06
Average	0.0005088567734

Performance Comparison

Matrix Type	**Avg Runtime (seconds) System #1**	**Avg Runtime (seconds) System #2**
Integer	0.01400358677	0.0211442709
Double	0.0007973194122	0.0005088567734

“Make sure your work includes a description of the two systems (manufacturer, CPU type, amount of memory, operating system, etc.) and of the compiler used (5).”

I used Python 3.9.7 for this assignment. I built my script in Atom and used the built-in Python compiler that is included with Python 3.9.7.

Spec	**System #1**	**System #2**
Manufacturer	iBUYPOWER	Apple
CPU Type	2.5 GHz Octa-Core Intel 11th Generation i7	2.8 GHz Quad-Core Intel Core i7
Memory (Installed RAM)	16 GB	16 GB
Operating System	Windows 10	macOS Big Sur

“Attach the source code, the tables with your time measurements for your work, and a link to your repository such that we can check-out the code, build, and execute (5).”

Please see attached 2a_matrix.py and 2b_matrix.py files for source code. My time measurement tables are included above in parts 2a and ab.

Link to repository on Github: https://github.com/jfontainedons/cs_402_hw1