Concurrency in bio-informatics. Dotplot.

Juan Ramírez

I. Introduction

DNA molecules are made of adenine (A), cytosine (C), guanine (G), and thymine (T), living organisms contain DNA, which consists of sequences of ACGT, in which is contained the genetic information of the living organism, the genome.[1]

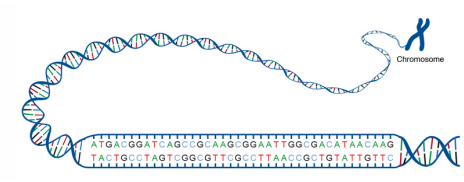

Fig. 1: [1]

What we want to achieve is to find the dotplot [2], which consists of comparing the DNA sequences of 2 organisms, and the result is a matrix that describes how far or how close are those organisms related.

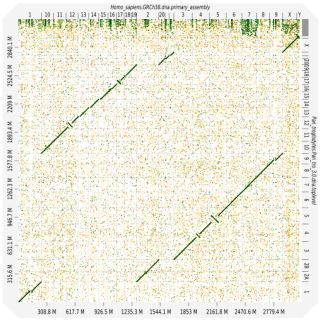

Fig. 2: [3]

Finding the dotplot is computationally expensive, what we want to do is to optimize the process using concurrency, parallelism and distributed computing.

II. Methodology

The language that we are going to work with is Python v3, and the libraries that we are going to work with are multiprocessing and mpi4py, these libraries enables us to work with parallelism and distributed computing.

The algorithmic complexity of finding the dotplot $O(n^2)$, because it requires to compare all the elements of a sequence A, against another sequence B, element by element.

$$\beta = \begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ f_n \end{array} \begin{array}{c} \begin{array}{cccc} \bar{f}_1 & \bar{f}_2 & \bar{f}_3 & \bar{f}_n \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & ... \\ 1 & 1 & 0 & ... \\ 0 & 0 & 0 & ... \\ ... & ... & ... & ... \end{pmatrix} \end{array}$$

β is the dotplot, f_n are the values ACGT of the sequence A and $\bar{f}_n$ are the values ACGT of the sequence B. Element by element they are compared and if they are equal a 1 is placed in that position, otherwise a 0.

Listing 1: Description of the sequential algorithm

```
import numpy as np

seqA = "ACGTGTC..."
seqB = "TGTACC..."

dotplot = np.zeros((len(seqA),
len(seqB)), dtype="bool")

dotplot = np.zeros((len(seqA),len(seqB)),
 dtype="bool")

for i in range(len(seqB)):
  for j in range(len(seqA)):
    if seqB[i] == seqA[j]:
      dotplot[j][i] = True

print(dotplot)
```

Running this algorithm on small strings is fast enough, but when the strings are big, the dotplot becomes bigger also, and it may not be able to be stored on RAM, also when the strings are big the process takes more time.

So that's why we are trying to implement concurrency and parallelism in this process, to be able to process big files.

One machine was used, and here are the specs.

RAM	5GB
Cores	10
Disk memory available	17GB

TABLE I: Specs of the computer that was used

III. Approach

Because the space in RAM is low, and the disk space is bigger, we are going to use this approach.

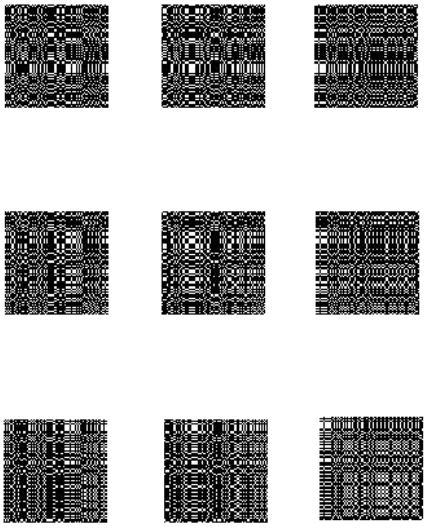

Fig. 3: Subplots, (the colors of the image were inverted for demonstration)

1) Make subplots and save each in an image, compressed and with a filter applied. As shown in Fig 3.
2) Join all the subplots into rows. As shown in Fig 4.

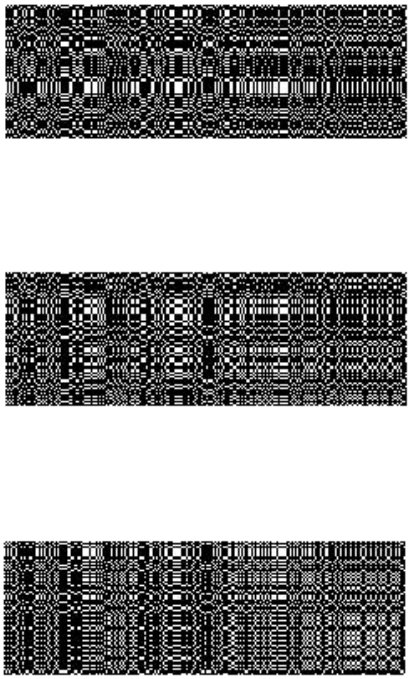

Fig. 4: Rows of subplots

3) Join all the rows, resulting in the final dotplot. As shown in Fig 5..

IV. Experiments and Results

We are going to compare 2 sequences E-coli and Salmonella.

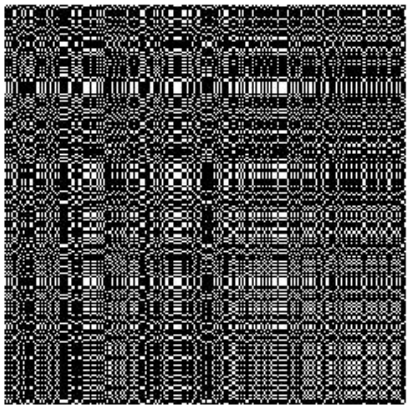

Fig. 5: Dotplot

E-coli has (58020*80) + 52 characters, so in total we have 4641652 characters, and Salmonella has 61894*80 + 13 characters, in total 4951533 characters.The resulting dotplot it's a matrix of with dimensions 4641652x4951533. If each element element of the dotplot matrix takes 1 bit, true or false, the size of the dotplot is 4641652x4951533 bits, or 2.6129 TB (TeraBytes). So if we want to run the sequential algorithm we need a machine with more than 2.6129 TB of RAM, and lets say that each comparison takes 0.001 seconds the approximated time of execution will be 4641652 *4951533 * 0.001 or 266010.33626 days. Using parallelism and distributed computing may reduce that execution time.

Listing 2: Implementation

```
"""
This code generates images used to make a dotplot given two dna sequences

This implementation requires that the 2 files have equal amount of lines and that
each line has the same amount of elements
(it has not been tested otherwise)
"""

import numpy as np
import matplotlib.pyplot as plt
import re
import time
from PIL import Image
import cv2
import argparse
import sys
import subprocess
from functionsShared import *
from multiprocessing import Pool

"""
clean images_subdotplot
"""
subprocess.call('sudo rm images_subdotplot/ -R', shell=True)
subprocess.call('mkdir images_subdotplot', shell=True)

"""
Constants
"""
SIZE = ""
CORES = 0

parser = argparse.ArgumentParser()

parser.add_argument("file1")
parser.add_argument("file2")
parser.add_argument("size")
parser.add_argument("cores")

args = parser.parse_args()
totaltime = time.time()
start = time.time()

try:
    file1 = readFile(args.file1)
    file2 = readFile(args.file2)
except:
    print("Verify that the names of the files are correct")
    sys.exit()

try:
```

```
    SIZE = int(args.size)
    """
    it returns the N elements, if the user wants to compare the first 100,
     or 200, or N value
    """
    file1 = file1[:SIZE]
    file2 = file2[:SIZE]
except:
    print("Verify that the lines size can be fitted into both
    sequences, remember it needs to be the same for both files")
    sys.exit()
try:
    CORES = int(args.cores)
except:
    print("In cores enter a number")
    sys.exit()
if CORES <= 0:
    print("In cores enter a number greater than zero")
    sys.exit()
"""
file1 and file2 has the same amount of lines and elements in each line
"""
writeToLog("number of cores {}".format(CORES))

writeToLog("lines size {}".format(SIZE))

pool = Pool(processes=CORES)

for i in range(len(file2)):
    for j in range(len(file1)):

        if len(file2[i]) == 1 or len(file1[j]) == 1:
            """
            dont compare strings with "\n" elements or only one element
            thats what that if does
            """
            break
        pool.apply_async(process,
        (processString(file2[i]),processString(file1[j]),i,j))
pool.close()
pool.join()

aux1 = time.time()
print("ellapsed making the subplots",aux1-start)
writeToLog("ellapsed making the subplots {}".format(aux1-start))

subprocess.run(["python3", "concat_images.py", str(SIZE),str(CORES)])

aux2 = time.time()
print("ellapsed total time",aux2-totaltime)
writeToLog("ellapsed total time {}".format(aux2-totaltime))

#file size of all the directory
res = subprocess.run(["du", "-h", "-s"],capture_output=True,text=True)

print("size of all the folder {}".format(res.stdout))
writeToLog("size of all the folder {}".format(res.stdout))
writeToLog("")
```

A. Data

The data is also found on the repository.
We have 2 files Salmonella.fna and E_coli.fna.
Here is the structure of one file.

```
1  AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGCTTCTGAACTG
2  GTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAATATAGGCATAGCGCACAGAC
3  AGATAAAAATTACAGAGTACACAACATCCATGAAACGCATTAGCACCACCATTACCACCACCATCACCATTACCACAGGT
4  AACGGTGCGGGCTGACGCGTACAGGAAACACAGAAAAAAGCCCGCACCTGACAGTGCGGGCTTTTTTTTTCGACCAAAGG
5  TAACGAGGTAACAACCATGCGAGTGTTGAAGTTCGGCGGTACATCAGTGGCAAATGCAGAACGTTTTCTGCGTGTTGCCG
6  ATATTCTGGAAAGCAATGCCAGGCAGGGGCAGGTGGCCACCGTCCTCTCTGCCCCCGCCAAAATCACCAACCACCTGGTG
7  GCGATGATTGAAAAAACCATTAGCGGCCAGGATGCTTTACCCAATATCAGCGATGCCGAACGTATTTTTGCCGAACTTTT
8  GACGGGACTCGCCGCCGCCCAGCCGGGGTTCCCGCTGGCGCAATTGAAAACTTTCGTCGATCAGGAATTTGCCCAAATAA
9  AACATGTCCTGCATGGCATTAGTTTGTTGGGGCAGTGCCCGGATAGCATCAACGCTGCGCTGATTTGCCGTGGCGAGAAA
10 ATGTCGATCGCCATTATGGCCGGCGTATTAGAAGCGCGCGGTCACAACGTTACTGTTATCGATCCGGTCGAAAAACTGCT
11 GGCAGTGGGGCATTACCTCGAATCTACCGTCGATATTGCTGAGTCCACCCGCCGTATTGCGGCAAGCCGCATTCCGGCTG
12 ATCACATGGTGCTGATGGCAGGTTTCACCGCCGGTAATGAAAAGGCGAACTGGTGGTGCTTGGACGCAACGGTTCCGAC
13 TACTCTGCTGCGGTGCTGGCTGCCTGTTTACGCGCCGATTGTTGCGAGATTTGGACGGACGTTGACGGGGTCTATACCTG
14 CGACCCGCGTCAGGTGCCCGATGCGAGGTTGTTGAAGTCGATGTCCTACCAGGAAGCGATGGAGCTTTCCTACTTCGGCG
15 CTAAAGTTCTTCACCCCCGCACCATTACCCCCATCGCCCAGTTCCAGATCCCTTGCCTGATTAAAAATACCGGAAATCCT
16 CAAGCACCAGGTACGCTCATTGGTGCCAGCCGTGATGAAGACGAATTACCGGTCAAGGGCATTTCCAATCTGAATAACAT
17 GGCAATGTTCAGCGTTTCTGGTCCGGGGATGAAAGGGATGGTCGGCATGGCGGCGCGCGTCTTTGCAGCGATGTCACGCG
18 CCCGTATTTCCGTGGTGCTGATTACGCAATCATCTTCCGAATACAGCATCAGTTTCTGCGTTCCACAAAGCGACTGTGTG
19 CGAGCTGAACGGGCAATGCAGGAAGAGTTCTACCTGGAACTGAAAGAAGGCTTACTGGAGCCGCTGGCAGTGACGGAACG
20 GCTGGCCATTATCTCGGTGGTAGGTGATGGTATGCGCACCTTGCGTGGGATCTCGGCGAAATTCTTTGCCGCACTGGCCC
21 GCGCCAATATCAACATTGTCGCCATTGCTCAGGGATCTTCTGAACGCTCAATCTCTGTCGTGGTAAATAACGATGATGCG
22 ACCACTGGCGTGCGCGTTACTCATCAGATGCTGTTCAATACCGATCAGGTTATCGAAGTGTTTGTGATTGGCGTCGGTGG
23 CGTTGGCGGGTGCGCTGCTGGAGCAACTGAAGCGTCAGCAAAGCTGGCTGAAGAATAAACATATCGACTTACGTGTCTGCG
24 GTGTTGCCAACTCGAAGGCTCTGCTCACCAATGTACATGGCCTTAATCTGGAAAACTGGCAGGAAGAACTGGCGCAAGCC
25 AAAGAGCCGTTTAATCTCGGGCGCTTAATTCGCCTCGTGAAAGAATATCATCTGCTGAACCCGGTCATTGTTGACTGCAC
26 TTCCAGCCAGGCAGTGGCGGATCAATATGCCGACTTCCTGCGCGAAGGTTTCCACGTTGTCACGCCGAACAAAAAGGCCA
27 ACACCTCGTCGATGGATTACTACCATCAGTTGCGTTATGCGGCGGAAAAATCGCGGCGTAAATTCCTCTATGACACCAAC
28 GTTGGGGCTGGATTACCGGTTATTGAGAACCTGCAAAATCTGCTCAATGCAGGTGATGAATTGATGAAGTTCTCCGGCAT
29 TCTTTCTGGTTCGCTTTCTTATATCTTCGGCAAGTTAGACGAAGGCATGAGTTTCTCCGAGGCGACCACGCTGGCGCGGG
30 AAATGGGTTATACCGAACCGGACCCGCGAGATGATCTTTCTGGTATGGATGTGGCGCGTAAACTATTGATTCTCGCTCGT
31 GAAACGGGACGTGAACTGGAGCTGGCGGATATTGAAATTGAACCTGTGCTGCCCGCAGAGTTTAACGCCGAGGGTGATGT
32 TGCCGCTTTTATGGCGAATCTGTCACAACTCGACGATCTCTTTGCCGCGCGCGTGGCGAAGGCCCGTGATGAAGGAAAAG
```

Fig. 6: Structure of E_coli.fna

The numbers that are on the left are the #lines. The user can change that value, this #lines makes the dotplot a matrix of #lines x #lines.

B. Warnings

The program compares both sequences and each of those need to have the same amount of lines to compare, it doesn't matter if one has 100 lines and the other 200 lines, the user can enter the value of the lines to compare, and if the user presses 100 lines, only 100 lines will be compared in the sequence1 and 100 in the sequence2 resulting in a square dotplot matrix, other dimensions are not supported.

The program was tested using linux.

C. How to run this program

Link to the repository:
To execute, enter into terminal and press

Listing 3: How to execute

```
python3 dotplot.py "seqA" "seqB" #lines #cores
```

Where "seqA" and "seqB" are DNA sequences in fna format, #lines is the lines to compare, and #cores is as its name implies.

D. Breakdowns

We are measuring

1) The time it takes to make all the subplots as shown in Fig.3
2) The time it takes to make all the rows using the subplots as shown in Fig.4
3) The time it takes to join all the rows, resulting in the final dotplot as shown in Fig.5
4) The total amount of time
5) The disk space that is being used

The breakdowns are printed to the user and are stored in a file called log.

E. Sequential results

To execute the sequential version, enter into terminal and press

Listing 4: How to execute sequential version

```
python3 dotplot.py "seqA" "seqB" #lines 1
```

In #cores enter 1.

We are going to run the program in 100, 200, 300 and 400 lines.

Listing 5: How to execute sequential version using 100 lines

```
python3 dotplot.py "seqA" "seqB" 100 1
```

F. Parallel results

1) Multiprocessing: Multiprocessing is a package that supports spawning processes using an API similar to the threading module. The multiprocessing package offers both local and remote concurrency, effectively side-stepping the Global Interpreter Lock by using subprocesses instead of threads. Due to this, the multiprocessing module allows the programmer to fully leverage multiple processors on a given machine.[4]

What we are going to do is, divide the process of the dotplot into subprocesses, and save to a file that processing, then join all those files together.

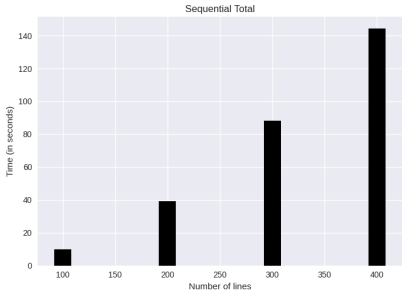

Fig. 7: Total sequential time

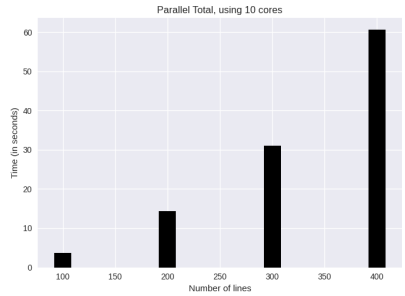

Fig. 8: Total parallel time, using 10 cores on each run

2) SpeedUp: The speedup S is described as Ts (sequential time) divided by Tp (parallel time).

$$S = \frac{Ts}{Tp}$$

#Lines	Speedup
100	2.69
200	2.73
300	2.84
400	2.38

TABLE II: Speedup

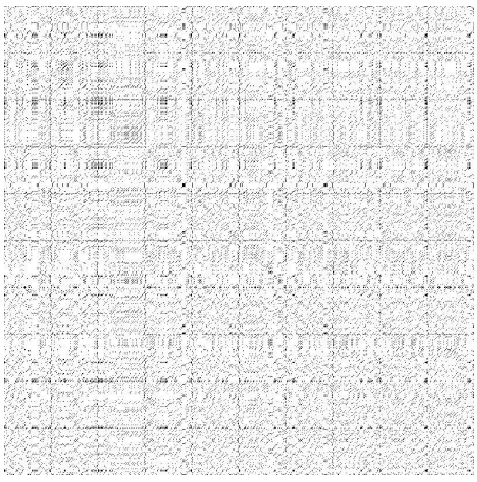

Fig. 9: Result of the dotplot

V. Repository

https://github.com/j17018/dotplot

VI. Further research

We think that doing the dotplot on a cluster with distributed computing may be the answer to the problem, the cluster would need enough memory to fit the size of the dotplot, that we think it should be around 2.6 TB.

VII. References

[1] ”The Human Genome Project”. genome.gov. Retrieved 3 June 2023.

[2] Gibbs, Adrian J.; McIntyre, George A. (1970). ”The Diagram, a Method for Comparing Sequences. Its Use with Amino Acid and Nucleotide Sequences”. Eur. J. Biochem. 16 (1): 1–11. doi:10.1111/j.1432-1033.1970.tb01046.x. PMID 5456129.

[3] Cabanettes F, Klopp C. (2018) D-GENIES: dot plot large genomes in an interactive, efficient and simple way.

[4]”Python Standard Library”. docs.python.org/3. Retrieved 3 June 2023.