COP5615 - Project 1

1. *Logistics*

a. Group Members

   i. Kaushik Manchenahalli Ranganatha Rao (6110-8000)

   ii. Sukhmeet Singh (1927-195)

b. How to run

   i. Unzip the file ManchenahalliRanganathaRao_Singh.zip

   ii. Navigate to supervisor/sequence in the project repo

   iii. Execute the command mix run proj1.exs N1 N2

   Ex: mix run proj1.exs 100000 200000

2. *Implementation Details*

Algorithm for Vampire Numbers

Given the range, start evaluating each number whether it is Vampire, using the below logic.

   i. Ignore all the odd length numbers and work on only the even length ones.

   ii. Convert the given number to string and sort the digits. Ex: if the number you are checking for is 1260, convert this to “0126” and store it in a variable strV

   iii. Look for fangs in the following range: the lower limit being the original number truncated to half and the upper limit being the square root of the number. If the fang is a factor of the number passed, store that as fang1 and compute fang2 using number/fang1

   iv. If the concatenated sorted string of fang1 and fang2 is the same as the initial string strV, **and** fang1 and fang2 do not both trail with 0

and if both the fangs are half the length, only then we say that we have found the fangs and the given number is a Vampire number.

v. Keep looking for more fangs in the range as described above.

Number of Worker Actors

Number of actors = Number of split ranges

3. Size of the work unit

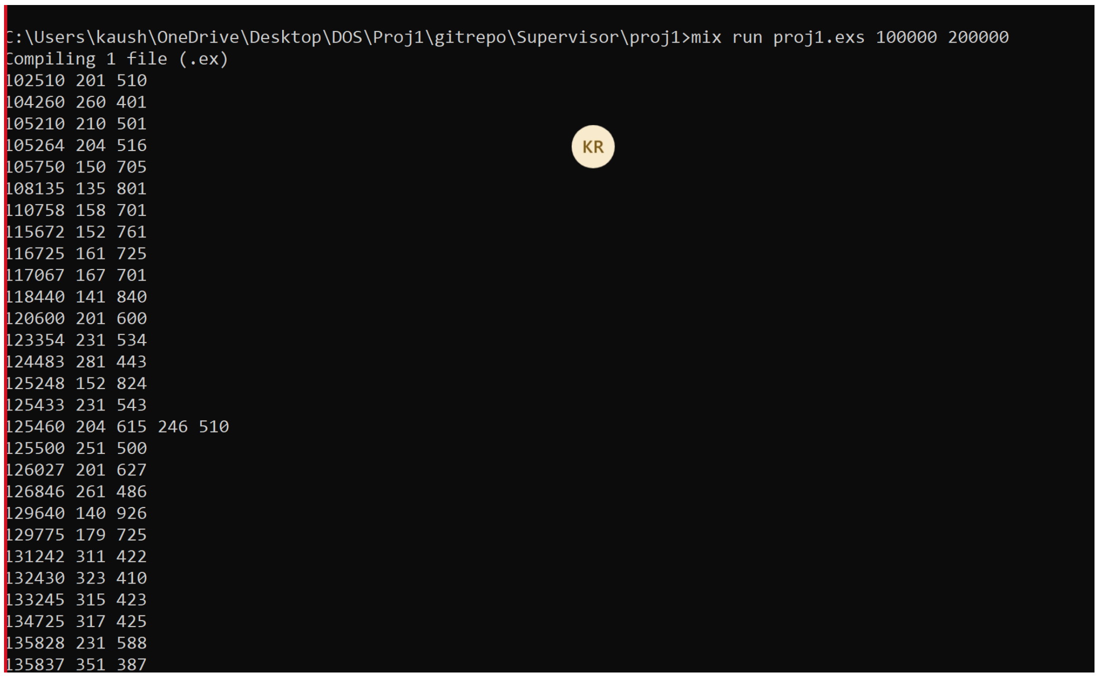
We are dividing the range among the actors and each actor is working on that range, so the size of work unit is the range divided by the number of cores. In a single request from the boss, each worker gets assigned a number between one and n for which it will compute the above formula. We approached this work size by doing an experiment. We incrementally spawned workers and calculated the time. For sample size of n1 = 1000, n2 = 2000, we observed as the number of workers increased the overall time taken to compute results decreased. However, there was little difference in the total time after 100 workers. Since, in Elixir actors have little overhead and in our optimized solution there is no state preservation required, we are spawning 'n' instances of the worker, each solving one sub-problem

4. Result of running n1 = 100000 and n2 = 200000

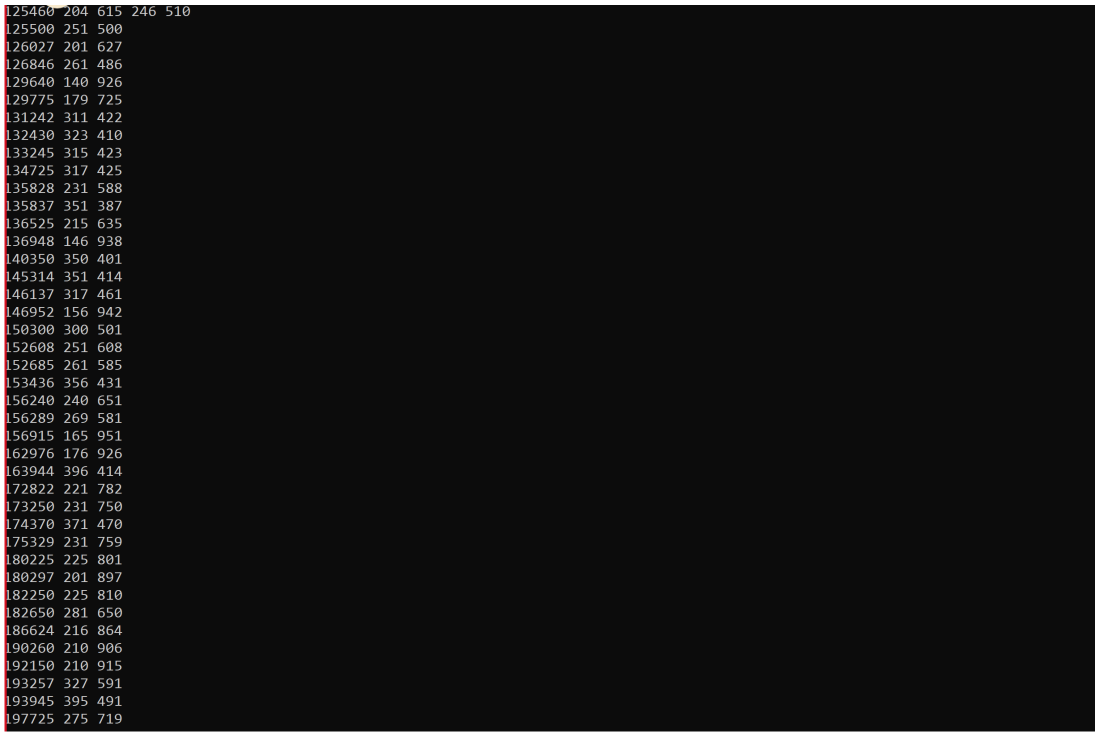
Real time: 0m2.371s

User Time: 0m8.422s

Sys Time: 0m0.969s



```
C:\Users\kaush\OneDrive\Desktop\DOS\Proj1\gitrepo\Supervisor\proj1>mix run proj1.exs 100000 200000
Compiling 1 file (.ex)
102510 201 510
104260 260 401
105210 210 501
105264 204 516
105750 150 705
108135 135 801
110758 158 701
115672 152 761
116725 161 725
117067 167 701
118440 141 840
120600 201 600
123354 231 534
124483 281 443
125248 152 824
125433 231 543
125460 204 615 246 510
125500 251 500
126027 201 627
126846 261 486
129640 140 926
129775 179 725
131242 311 422
132430 323 410
133245 315 423
134725 317 425
135828 231 588
135837 351 387
```



```
125460 204 615 246 510
125500 251 500
126027 201 627
126846 261 486
129640 140 926
129775 179 725
131242 311 422
132430 323 410
133245 315 423
134725 317 425
135828 231 588
135837 351 387
136525 215 635
136948 146 938
140350 350 401
145314 351 414
146137 317 461
146952 156 942
150300 300 501
152608 251 608
152685 261 585
153436 356 431
156240 240 651
156289 269 581
156915 165 951
162976 176 926
163944 396 414
172822 221 782
173250 231 750
174370 371 470
175329 231 759
180225 225 801
180297 201 897
182250 225 810
182650 281 650
186624 216 864
190260 210 906
192150 210 915
193257 327 591
193945 395 491
197725 275 719
```

5. *CPU Utilization Ratio of CPU time to Real Time*

Total CPU Time = 9.391s

Total Real Time = 2.371s

The Ratio of the CPU time to Real time = 3.96

6. ***Largest Problem we solved***

For a single machine, we could get vampire numbers in the range 10000000 to 11000000

Real time: 3m36.575s

User time: 10m15.109s

Sys time: 0m45.984s

Ratio of CPU time to Real Time ~ 12

Using multiple machines, we could get vampire numbers in the range 1000000000 to 2000000000