Project 1 - COP 5615

Group members:

- Oscar Camargo
- Lakksh Tyagi

The results of our project were gathered from running the program on a MacBook Air, 2020, Apple M1, 16GB

For our implementation, we have four types of actors: Main, Computer, Checker, Displayer. Main has the role of creating the actors and starting the Computer actors. Displayer has the role of displaying the results that Checker has determined satisfy the requirements to be a solution to the problem. There is only one Displayer and one Main actor. The Checker and Computers actors are created depending on the size of n, since our solution is independent of k for its time complexity. No matter the size of n, the Computers actor sends an array of size 10,000 (F64) to Checker with what they have calculated. This communication is made that way to reduce message passing. We determined that the best work size for each actor is to have them perform operations to at most 10,000 F64 numbers. This was determined through benchmarking. However, further experiments can be done. For example, changing both the formula to get the number of Checker and Computer actors, and changing the size of the arrays simultaneously. I think this experiment may lead to other ways of improving the program since changing both the number of actors and the size of their workload could affect the runtime. Another example is making the number of actors constant and varying the size of their work unit. Furthermore, as n increases since what varies is the number of actors, the performance suffers, so this modification could prove beneficial. However, what we truly optimized was the solution for the given test cases (including the extra credit), so we are satisfied with its current performance for that range of numbers.

System reported: ./Lukas 1000000 4 0.05s user 0.01s system 356% cpu 0.015 total

```
(base) lakkshtyagi@Lakkshs-MacBook-Air Pony % time ./Lukas 1000000 4
./Lukas 1000000 4  0.05s user 0.01s system 356% cpu 0.015 total
```

Answer: No output

Thus, cores (ratio) = (0.05+0.01)/(0.015) = 4

The largest problem, with a solution, that we managed to solve correctly was for n=10000000 and k=16346.

System reported: ./lukas 10000000 16346 0.29s user 0.03s system 631% cpu 0.051 total

```
(base) lakkshtyagi@Lakkshs-MacBook-Air Pony % time ./Lukas 10000000 16346
138717
./Lukas 10000000 16346  0.29s user 0.03s system 631% cpu 0.051 total
```

Answer: 138717

We checked this answer with a Python script and to choose valid k's we used the following webpage as a reference: https://oeis.org/A001032#:~:text=When%20n%20is%20not%20a%20square%2C%20the%20solution,terms%20is%2025%5E2%20%2B...%20%2B%2073%5E2%20%3D%20357%5E2

This is the largest problem, with a solution, that we found that we managed to solve correctly. Some reasons we believe we were not able to solve larger problems are that the way we determine whether a number is a perfect square is an approximation; besides, the numbers increase at an exponential rate, so there may be issues with our floating point numbers. A way to solve this could be checking that we did not pass the upper bound or using better numerical methods to compute a solution.

The largest problem, with no solution or that yielded no output, that we managed to solve was for n = 1000000000 and k = 230000.

System reported: ./lukas 1000000000 230000 8.94s user 0.27s system 702% cpu 1.311 total

```
(base) oscarcamargo@Oscars-MacBook-Air lukas % time ./lukas 1000000000 230000
./lukas 1000000000 230000  8.94s user 0.27s system 702% cpu 1.311 total
```

As mentioned before, our solution is independent of k. However, increasing n by another factor of 10 makes our system really slow. This may be due to the significant increase in the number of actors that are created.

Finally, our program prints the solutions to the problem with any order. We decided to implement it this way so that the printing could also be done concurrently instead of sequentially.

Bonus (15%)

System reported: ./Lukas 100000000 20 2.35s user 0.22s system 550% cpu 0.467 total

```
(base) lakkshtyagi@Lakkshs-MacBook-Air Pony % time ./Lukas 100000000 20
8.82445e+07
3.24089e+07
3.55032e+07
8.60097e+07
5.64932e+07
9.78284e+07
8.15402e+07
4.67976e+07
3.11798e+07
8.74367e+07
2.55326e+07
9.12358e+07
7.25149e+07
9.38314e+07
3.89328e+07
9.23788e+07
1.68383e+07
6.56989e+07
8.46688e+07
5.72754e+07
2.90739e+07
7.13975e+07
6.18567e+07
7.98897e+07
6.72632e+07
7.56867e+07
./Lukas 100000000 20  2.35s user 0.22s system 550% cpu 0.467 total
```

Thus, cores (ratio) = (2.35+0.22)/(0.467) = 5.5

As shown above, our solution can solve the extra credit test case on a single multicore machine.

Our research indicated that Pony does not support remote actors, so our solution would involve connecting two machines through a TCP connection. Have one of the machines deal with the workload of checking and displaying and the other with the task of computing. To send the arrays through the TCP stream, we would then have to serialize and deserialize them.