Project 1: Sums of Consecutive Squares

Distributed Operating Systems Principles - Fall 2025

1 Problem Definition

An interesting problem in arithmetic with deep implications to elliptic curve theory is the problem of finding perfect squares that are sums of consecutive squares. A classic example is the Pythagorean identity:

$$3^2 + 4^2 = 5^2 \tag{1}$$

This reveals that the sum of squares of 3 & 4 is itself a square. A more interesting example is Lucas' Square Pyramid:

$$1^2 + 2^2 + ... + 24^2 = 70^2 \tag{2}$$

In both of these examples, sums of squares of consecutive integers form the square of another integer. The goal of this first project is to use **Gleam** and the actor model to build a good solution to this problem that runs well on multi-core machines.

2 Requirements

Input: The input provided (as command line to your program, e.g. `lukas`) will be two numbers: N and k. The overall goal of your program is to find all k consecutive numbers starting at 1 or higher, and up to N, such that the sum of squares is itself a perfect square (of an integer).

Output: Print, on independent lines, the first number in the sequence for each solution.

Example 1:

```
lukas 3 2
3
```

indicates that sequences of length 2 with start point between 1 and 3 contain 3, 4 as a solution since $3^2 + 4^2 = 5^2$.

Example 2:

```
lukas 40 24
1
```

indicates that sequences of length 24 with start point between 1 and 40 contain 1, 2 , ..., 24 as a solution since $1^2 + 2^2 + ... + 24^2 = 70^2$.

2.1 Actor modeling

In this project you have to use exclusively the actor facility in **Gleam** *(projects that do not use multiple actors or use any other formof parallelism will receive no credit).*

A model similar to the one indicated in class for the problem of adding up a lot of numbers can be used here, in particular define worker actors that are given a range of problems to solve and a boss that keeps track of all the problems and perform the job assignment.

2.2 README File

In the README file you have to include the following material:

- The size of the work unit that you determined results in best performance for your implementation and an explanation on how you determined it.
  - A size of the work unit refers to the number of sub-problems that a worker gets in a single request from the boss.
- The result of running your program for `lukas 1000000 4`
- The **REAL TIME** as well as the ratio of **CPU TIME** to **REAL TIME** for the above, i.e. for `lukas 1000000 4`.
  - The ratio of **CPU TIME** to **REAL TIME** tells you how many cores were effectively used in the computation.
  - If your ratio is close to 1, you have almost no parallelism and points will be subtracted.
- The largest problem you managed to solve.

3 BONUS – 15%

Use remote actors and run your program on 2+ machines. Use your solution to solve a large instance such as: `lukas 100000000 20`. **To receive bonus points you must record a video demo and explain your solution.**