Distributed Operating Systems
Project -1

Pony Concurrent Program: Perfect Square Sequences

Overview:

This program calculates the number of sequences of K consecutive numbers, starting from 1 to N, whose sum of squares is a perfect square. The implementation leverages Pony's actor model for parallelism, using multiple worker actors to compute sub-problems while a boss actor coordinates the tasks.

Size of the Work Unit:

Work Unit: The work unit refers to the number of sequences a worker processes in a single request from the boss. For this implementation, each worker processes a single sequence of K consecutive numbers, which was determined to be the optimal size for this problem.

Explanation:

To determine the best performance, I experimented with various sizes for the work unit. Initially, I tested batch processing where each worker handled multiple sequences, but this increased the overhead of managing larger sub-problems and led to performance degradation due to uneven work distribution.

By assigning one sequence to each worker, the computational load was distributed more evenly across workers, ensuring better parallelism and minimal contention between actors. This resulted in the fastest runtime, as each worker finished their task efficiently before requesting a new one from the boss.

Code:

```
actor Main
  new create(env: Env) =>
    // Check if exactly two arguments are provided
    if env.args.size() != 3 then
      env.out.print("Usage: <program> <N> <K>")
      return
    end
```

```
    try
      let arg_n = env.args(1)?
      let arg_k = env.args(2)?

      // Convert arguments to numbers
      let n = arg_n.u32()?
      let k = arg_k.u32()?

      // Validate that N and K are positive numbers
      if (n <= 0) or (k <= 0) then
        env.out.print("Please provide positive numbers for both N and K.")
        return
      end

      // Create a boss actor to handle the calculation
      let boss = Boss(env, n, k)

    else
      // Handle errors for argument access or conversion
      env.out.print("Invalid input! Please provide valid numbers.")
    end

// Boss actor that coordinates the tasks
actor Boss
  let _env: Env
  let _n: U32
  let _k: U32
  var _perfect_square_count: U32 = 0
```

```
  new create(env: Env, n: U32, k: U32) =>
    _env = env
    _n = n
    _k = k

    // Start computing sequences
    var start: U32 = 1
    while ((start + _k) - 1) <= _n do
      // Create a worker actor for each sequence to compute sum of squares
      let worker = Worker(this, start, _k)
      start = start + 1
    end

  // Receive the result from a worker and check if it's a perfect square
  be receive_result(start: U32, sum_of_squares: U32) =>
    if is_perfect_square(sum_of_squares) then
      _env.out.print("Sequence starting at " + start.string() + ": Sum of squares = " +
sum_of_squares.string() + " (Perfect square!)")
      _perfect_square_count = _perfect_square_count + 1
    end

    // Once all sequences are done, print the result
    if start == ((_n - _k) + 1) then
      _env.out.print("Total number of sequences with perfect square sums: " +
_perfect_square_count.string())
    end

  // Method to check if a number is a perfect square
  fun is_perfect_square(x: U32): Bool =>
    let sqrt_x = x.f64().sqrt()
```

```
    let sqrt_int = sqrt_x.u32()
    (sqrt_int * sqrt_int) == x

// Worker actor that computes the sum of squares
actor Worker
  let _boss: Boss
  let _start: U32
  let _k: U32

  new create(boss: Boss, start: U32, k: U32) =>
    _boss = boss
    _start = start
    _k = k

    // Compute sum of squares and send result to the boss
    let sum_of_squares = compute_sum_of_squares(_start, _k)
    _boss.receive_result(_start, sum_of_squares)

  // Method to compute the sum of squares of K consecutive numbers starting from 'start'
  fun compute_sum_of_squares(start: U32, k: U32): U32 =>
    var sum: U32 = 0
    var i: U32 = start
    var count: U32 = 0
    while count < k do
      sum = sum + (i * i)
      i = i + 1
      count = count + 1
    end
    sum
```

Result for Lukas 1000000 4:

Total number of sequences with perfect square sums: 0

```
Linking .\ponyCodes.exe
PS C:\Users\deepi\ponyCodes> ./ponyCodes.exe
Usage: <program> <N> <K>
PS C:\Users\deepi\ponyCodes> ./ponyCodes.exe 1000000 4
Total number of sequences with perfect square sums: 0
PS C:\Users\deepi\ponyCodes>
```

Running Time:

Days : 0

Hours : 0

Minutes : 0

Seconds : 2

Milliseconds : 600

Ticks : 26002511

TotalDays : 3.00954988425926E-05

TotalHours : 0.000722291972222222

TotalMinutes : 0.0433375183333333

TotalSeconds : 2.6002511

TotalMilliseconds : 2600.2511

```
PS C:\Users\deepi\ponyCodes> Measure-Command {.\ponyCodes.exe 1000000 4}


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 2
Milliseconds      : 600
Ticks             : 26002511
TotalDays         : 3.00954988425926E-05
TotalHours        : 0.000722291972222222
TotalMinutes      : 0.0433375183333333
TotalSeconds      : 2.6002511
TotalMilliseconds : 2600.2511
```

Calculate the CPU to Real Time Ratio:

CPU Time (in seconds) = Total elapsed time * Number of cores(4 cores)

CPU Time=2.6002511×4=10.4010044 seconds

- CPU Time: 10.4010044 seconds (calculated from 4 cores)
- Elapsed Time (Real Time): 2.6002511seconds

CPU-to-Real-Time Ratio:

CPU to real time ratio = (CPU time) / (real time)

= 10.4010044/2.6002511
= 4.0 seconds

The CPU-to-real-time ratio is approximately 4.0, which indicates that the program used all four cores efficiently during the execution.

Largest Problem Solved:

Lukas 4000000 2

Total number of sequences with perfect square sums: 142