Project 1 README
(For Local Actors Only)

Group Members:
Shyamala Athaide (UF-ID 68741819)
Neeraj Rao (UF-ID 37737151)

How to launch the program

1. Unzip the project zip files.
2. Compile the source from the root of the project directory using the command:
```
scalac –d out src/project1/project1.scala
```
3. Next, from the same directory, run the program and provide 2 arguments (1st argument is N, 2nd argument is K), using the command:
```
scala –cp out project1.Boss 100 24
```

Logic of the program

We have to calculate the sum of squares of K numbers starting from [1,N]. The most time consuming operation here is calculating the squares of the numbers. Hence, a naïve implementation that simply assigns number sequences to different actors would be very inefficient since the squares of numbers common to overlapping sequences would be calculated multiple times. As an example, consider a sequence length of K = 4. The first four sequences would be
1,2,3,4
2,3,4,5
3,4,5,6
4,5,6,7
We can see that the square of 2 would have to be calculated twice, the square of 3 thrice and the squares of the successive numbers (up to N-2) 4 times.

In order to avoid repetition, we use the fact that, in general, the sum of the squares of K numbers can be represented by
$$\left(M-\frac{K}{2}\right)^2+\ldots+(M-2)^2+(M-1)^2+(M)^2+(M+1)^2(M+2)^2+\ldots+\left(M+\frac{K}{2}\right)^2,$$ where M is the median of the sequence. This sum can be simplified to $K*M^2+2*C$, where
$$C=\left(1^2+2^2+\ldots+\left(\left\lfloor\frac{K}{2}\right\rfloor\right)^2\right)$$ for odd Ks
$$C=\left(0.5^2+1.5^2+\ldots+\left(\frac{K}{2}-0.5\right)^2\right)$$ for even Ks.
The advantages of this approach are that

a. Each sequence of K numbers is characterized by a *unique* median M and hence, to calculate the sum of each sequence, *only* this one number needs to be squared (i.e., we do not need to square the numbers common to the various sequences repeatedly)
b. C is the *same* for all sequences which means that it (and the squares that make it up) needs to be computed only once!

The constant term C and the range of medians M for a given K are shown in Table 1.

K	1st Median	Intermediate Medians	Last Median	C
Even	$\left(\frac{K}{2}+0.5\right)$	Add 1 to predecessor	$N+\left(\frac{K}{2}-0.5\right)$	$\left(0.5^2+1.5^2+\ldots+\left(\frac{K}{2}-0.5\right)^2\right)$
Odd	$\left\lceil\frac{K}{2}\right\rceil$		$N+\left\lfloor\frac{K}{2}\right\rfloor$	$\left(1^2+2^2+\ldots+\left(\left\lfloor\frac{K}{2}\right\rfloor\right)^2\right)$

Table 1: C and range of M for a given K

Program Architecture

The top level class Boss is given the N and K values. There is one layer of actors beneath him. In this layer, we have W units of class CCalculator and U units of class MCalculator. W and U are determined experimentally as described in the next section.

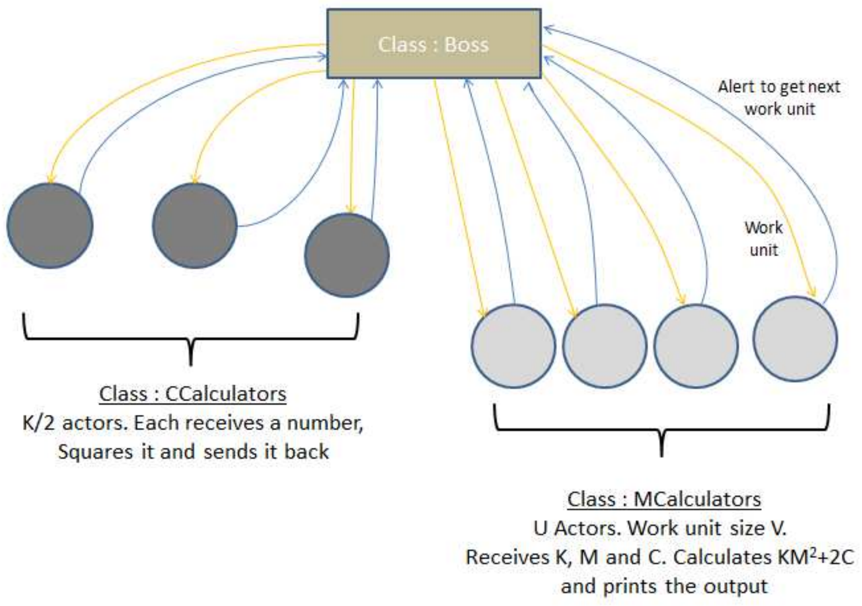

Class : Boss
Alert to get next work unit
Work unit
Class : CCalculators
K/2 actors. Each receives a number, Squares it and sends it back
Class : MCalculators
U Actors. Work unit size V.
Receives K, M and C. Calculates KM2+2C
and prints the output

Figure 1: Program Architecture

Since C remains constant given a K, we start by calculating C. As can be seen from Table 1, each of the W units is given either 1, 2 etc. if K is odd or 0.5, 1.5 etc. if K is even. The W units work in parallel and square the numbers that they are given. The squares are returned to the Boss, who adds them up to get C (this exchange of numbers is shown by the arrows between the Boss and CCalculator units). Once C has been calculated, these W units are killed. Note that if W is smaller than the minimum number of units required ($\frac{K}{2}$), the Boss reuses the units that have sent a response.

Once C has been calculated, we can proceed to get the sums of squares of the K-length sequences. This is done by the U units of class MCalculator. Each unit is given the median M and the value C that was calculated in the last step. Each unit then computes the sum *of the following V sequences* using $K * M^2 + 2 * C$ (where M increases by 1 for each successive sequence) and checks if the sum is a perfect square. If it is, the unit then prints the starting value of the sequence using the formula $\left(M - \frac{K}{2}\right).toInt + 1$. The arrows between MCalculator and Boss show that the Boss gives the starting median M for V consecutive sequences and the MCalculators respond asking for the next M when they're done. Note that if U is smaller than the minimum number of units required ($\frac{N}{V}$), the Boss reuses the units that have sent a response. Once all the medians have been dealt with, the Boss kills off the MCalculator units and then exits itself.

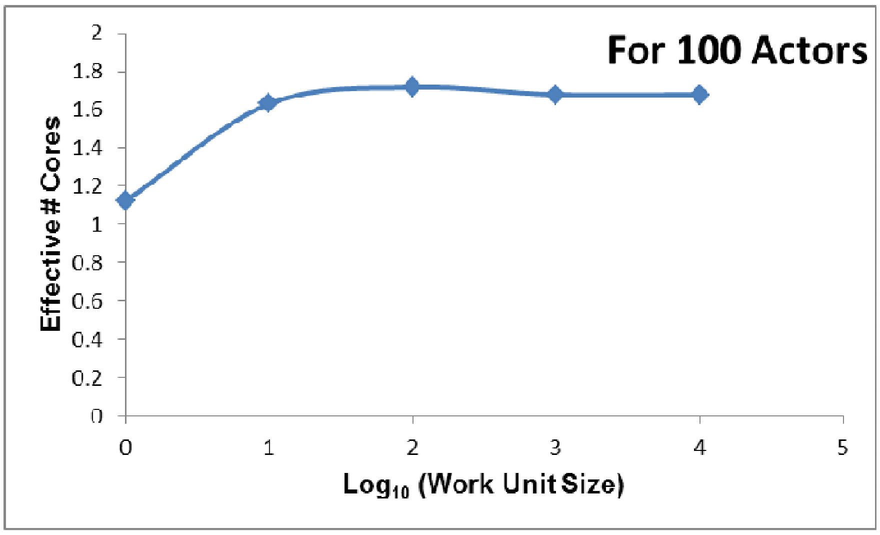
Size of work unit for best performance

To determine the best work unit size, V, we ran the program multiple times with various values of V and looked at the ratio of CPU time (kernel-mode + userspace mode) to Real time. The higher the ratio (obviously limited by the number of cores on the machine), the more parallelism achieved.

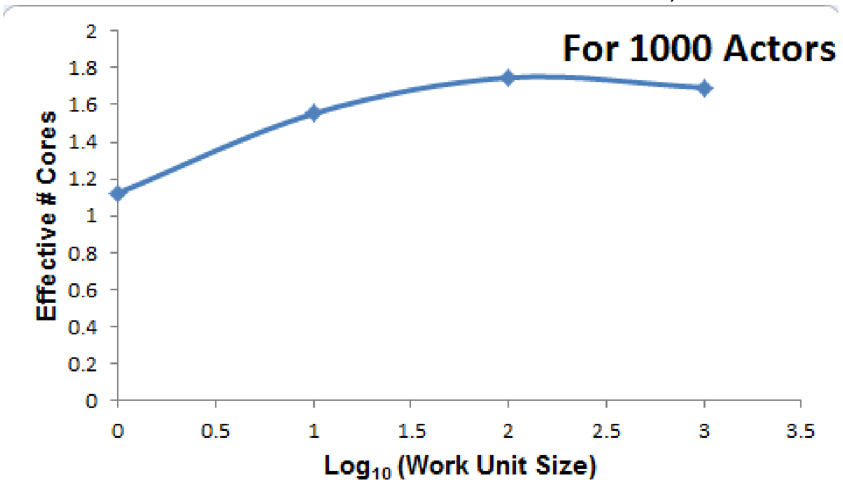
We also needed to characterize the network topology in terms of W and U. We use the same method as for determining V (running different combinations of W, U and looking at the run times).
For V, the value of work unit size sent per request, we ran different combinations of W, U and V. For recorded these values for a few high combinations of N and K. Our observation was as follows:

N=1000000, K=4

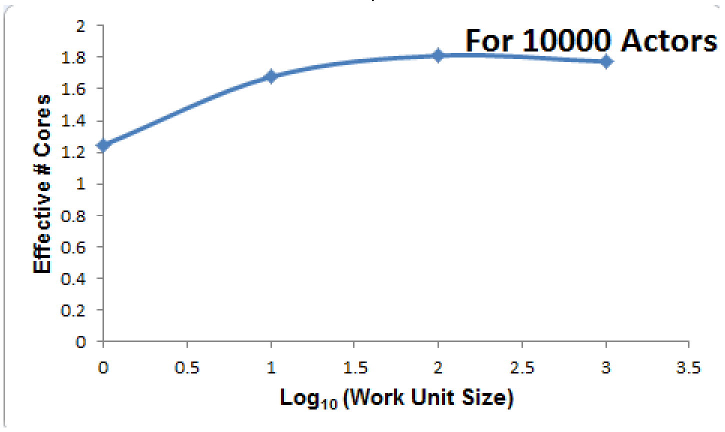
For U=100, CPU time vs. Real time ratio was 1.716 as shown below.



For U=1000, we achieve a CPU time to Real time ratio of 1.747 for V = 100, as shown in the graph below.



For U=10000, this ratio increases to 1.812 for V = 100, as shown below.



Thus, we achieved the most optimal CPU time to Real time ratio with a work unit size of V=100. With changing values of N we determined that if we set V as $N^{\frac{1}{3}}$ we achieved a ratio of close to 1.8 most cases. This was determined from the plots of Effective Cores Vs Log(V). For all the cases with different number of actors, the Curve attained its peak when Log(V) was around 1/3(Log(N))

Output of scala project1.scala 1000000 4

No sequences found that satisfy the condition.

Runtime for scala project1.scala 1000000 4

We observed that the optimal value of the number of actors for calculating the constant(C) was $W = \frac{K}{2}$ and $W = \frac{K}{4}$ interchangeably. However, smaller values of N along with $W = \frac{K}{2}$ yielded the highest ratio. We fixed the value of W to be $\frac{K}{2}$ to account for a varied range of N. This can be explained by the fact that we are squaring K/2 numbers to compute the value of C, so W = K/2 makes sure that all these computations occur in parallel and so C is computed in the least time. Additionally, since C is computed only once, these actors won't be alive for the whole duration of the program.

We also observed that for most high values of N, number of actors to calculate the medians, U, yielded the highest ratio when $U = N^{\frac{2}{3}}$.

This highest CPU time to Real time ratio we could achieve from our code is 1.81(CPU time = 6.94 s & Real time = 3.83 s) on a machine with 2 cores.

Largest problem solved

The largest problem we could solve without the system throwing an exception was for N = 1000000 with a CPU time to Real time ratio of 1.81 on a 2 core machine.

The code also ran for N=10000000 and N=100000000 with a CPU time to Real time ratio of around 1.7. However, these large values of N would often throw 'out of memory' exceptions (even after increasing JVM heap size using –Xmx and –Xms).