Group Members: Rajath A Ganesh [UFID : 5314-1354] , Prajwala Nagaraj [UFID : 1099-2662]

Project 1

The goal of this project is to use F# and the actor model to build a good solution to sum of squares problem that runs well on multi-core machines.

Prerequisites:

- .NET SDK latest
- F#
- Nuget

How to run

```
dotnet fsi --langversion:preview proj1.fsx 3 2
3
```

Size of Work Unit:

To calculate the sum of squares of a sequence 1..N, we use the following equation:

```
SUM(N) = N(N+1)(2N+1)/6
```

Therefore, for a sequence of *k* elements starting from *i*, the sum of squares will be:

```
SUM(i + k - 1) - SUM(i - 1)
```

That means each sequence only takes O(1).

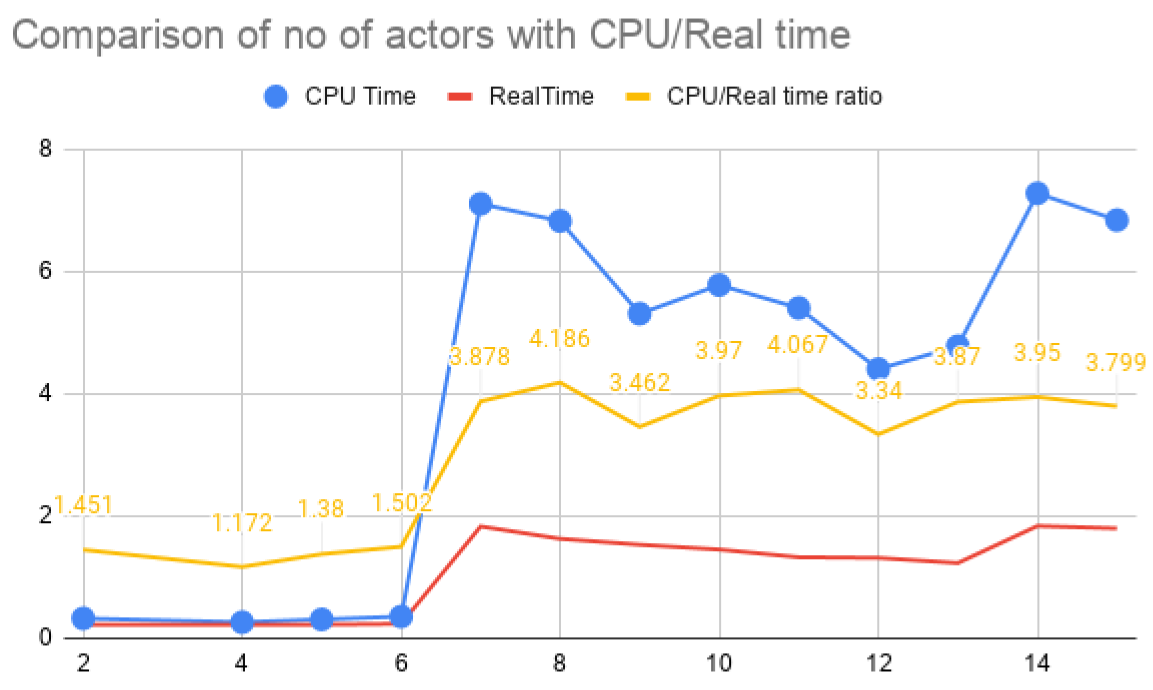
Denoting *c* as the number of workers, each one will be given a work unit of **(n/c)** sequence(s).
In case *n* is not divisible by *c*, the first **(n%c)** workers will be given 1 more sequence.

Example: with n = 10, we have 10 sequences. Suppose 4 workers are spawn, each worker will be assigned 3, 3, 2, and 2 sequences, respectively.

The number of actors can be changed by assigning the value at line at proj1.fsx:

```
7 // Work Unit - change as required to improve performance
```

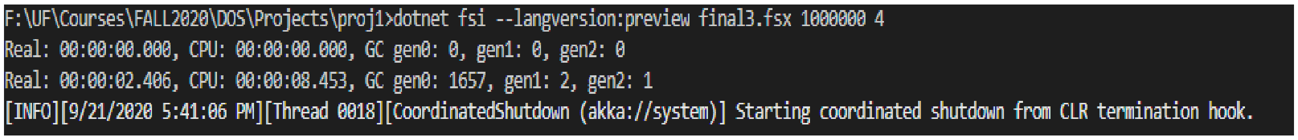
```
8 let numberOfActors = 10
```
To determine the size of the work unit we used the input `10^6 and k = 4` and used a range of number of units and we plotted the following graph:



By trying out different example inputs, we found the work unit at `2 * k` gives maximum CPU/Real time ratio.

Result:

The result for the problem asked (1000000 4) with 8 workers is given as :

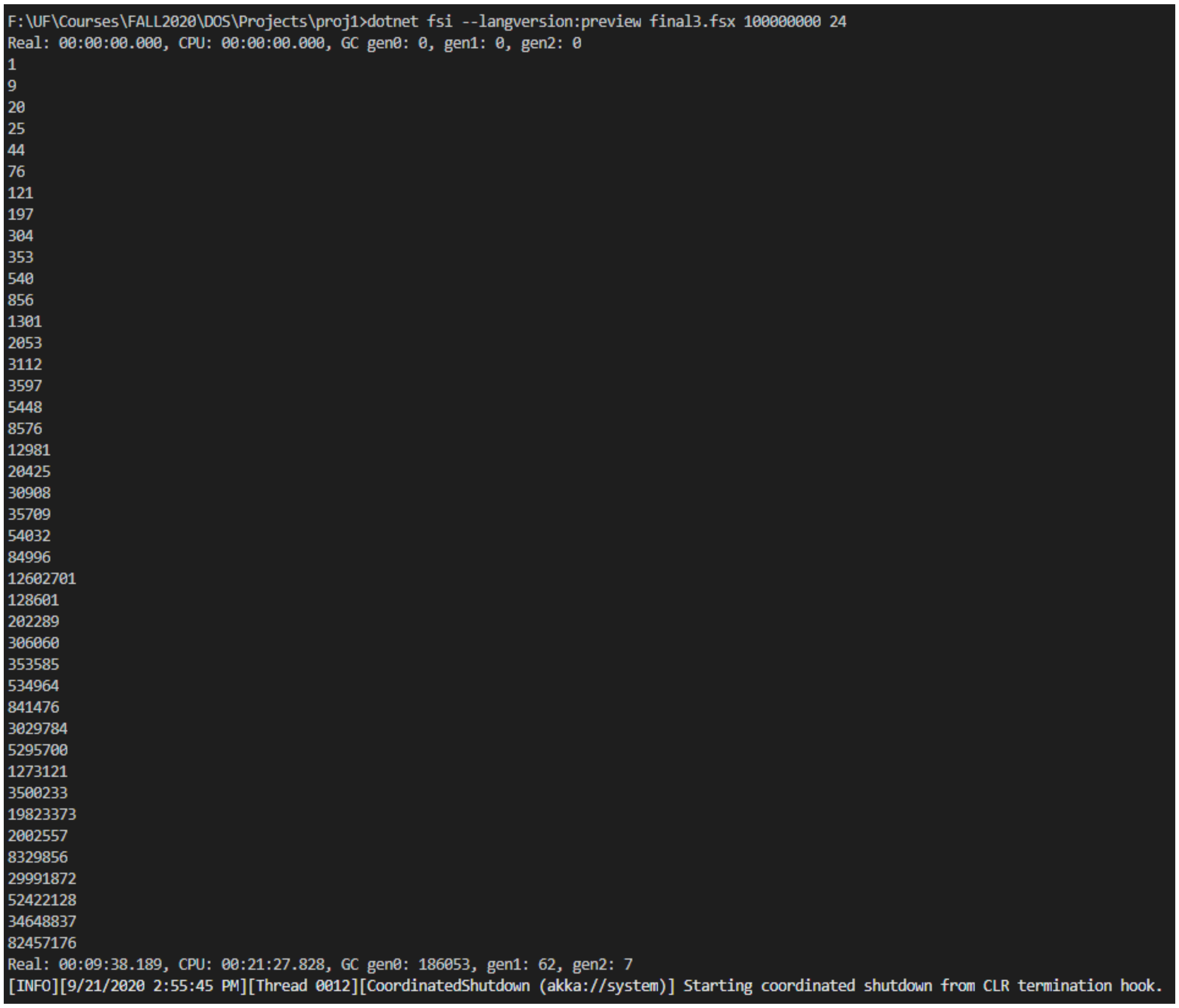

```
F:\UF\Courses\FALL2020\DOS\Projects\proj1>dotnet fsi --langversion:preview final3.fsx 1000000 4
Real: 00:00:00.000, CPU: 00:00:00.000, GC gen0: 0, gen1: 0, gen2: 0
Real: 00:00:02.406, CPU: 00:00:08.453, GC gen0: 1657, gen1: 2, gen2: 1
[INFO][9/21/2020 5:41:06 PM][Thread 0018][CoordinatedShutdown (akka://system)] Starting coordinated shutdown from CLR termination hook.
```

Running time:

On basis of input: 1000000 4 on 8 workers, the ratio (CPU/Real) we are getting is : 4.186

Largest problem:

The largest problem we tried was 100000000 24 with output as shown below:



```
F:\UF\Courses\FALL2020\DOS\Projects\proj1>dotnet fsi --langversion:preview final3.fsx 100000000 24
Real: 00:00:00.000, CPU: 00:00:00.000, GC gen0: 0, gen1: 0, gen2: 0
1
9
20
25
44
76
121
197
304
353
540
856
1301
2053
3112
3597
5448
8576
12981
20425
30908
35709
54032
84996
12602701
128601
202289
306060
353585
534964
841476
3029784
5295700
1273121
3500233
19823373
2002557
8329856
29991872
52422128
34648837
82457176
Real: 00:09:38.189, CPU: 00:21:27.828, GC gen0: 186053, gen1: 62, gen2: 7
[INFO][9/21/2020 2:55:45 PM][Thread 0012][CoordinatedShutdown (akka://system)] Starting coordinated shutdown from CLR termination hook.
```

Other Approaches tried

We also tried another approach where we calculated the squares and then summed them up using an actor and all ranges were given to different worked actors. The following were the hierarchy of actors:

- Master - Splits into a sliding window range.
- Processor - Spawns actors to calculate the squares of each number and then sums it up and checks if it is a perfect square and returns to the master
- Worker - Replies with the square of the given number. This approach took a lot of time complete. The time taken was 10 times more than the time in the approach done above. The file `Proj1WithoutFormula.fsx` has the code for the following approach and takes the same inputs.

Sequential Program for Testing

We created a sequential program (without actors) to calculate the same problem sequentially in F# to help us test our program. The file is called `normalF#Prog.fsx.`