# README

Jingyin Tang (8911-0469) Dawei Jia (9136-5931)

September 6, 2013

## 1 Compiling Source Codes

In this submission, complied class of "workers" and "boss" are contained. To compile from source code, please refer to "run.sh".

```
$ scalac boss.scala worker.scala
$ export CLASSPATH=$CLASSPATH:.
$ scala project1.scala
```

## 2 Size Of Worker Unit

The testing platform of this project is a laptop with one Intel i3 CPU at 1.1GHz and 4GB memory. To test an optimized size for a sub-problem, additional two parameter are added to the CLI.

```
$ scala project1.scala N k size count_of_worker
```

As one Intel i3 CPU has 2 physical cores and 4 logical cores, this program would not benefit from running more than 4 worker actors. In this project, scenario of 2 worker actors, 3 worker actors and 4 worker actors are tested under different size of worker unit. Two conclusion are made through multiple tests where N = [1E+4,1E+7], k=4

1. Since in this program, the sum function is resolved as

$$\sum_{i=n}^{n+k-1} i^2 = \sum_{i=1}^{n+k-1} i^2 - \sum_{i=1}^{n-1} i^2 = \frac{(n+k-1)\cdot(n+k)\cdot(2n+2k-1) - n\cdot(n-1)\cdot(2n-1)}{6}$$

   Increasing k would not increase time of computing.

2. 3 and 4 worker actors run somewhat faster than 2 workers which indicates this program could benefits from additional logical cores. However, 4 worker actors don't double the speed of 2 actors and there are no significant difference between 3 and 4 worker actors. So all further runnings on this platform will use 4 worker actors.

3. The size at around 25% of total workset gives best performance on this platform. Variants could be due to characters of logical processor. But, this result indicates that for this question, minimal messages to assign job to each core would gain the best performance.

## 3 Sample Running

```
$ scala project1.scala 1000000 4
```

In fact, this running gives no results. Here is the justification:
for any integer n satisfies 1 < n < N+2. Expected result is

$$(n-1)^2 + n^2 + (n+1)^2 + (n+2)^2 = 4n^2 + 4n + 6 = (2n+1)^2 + 5 = m^2$$

These is no such interger m exists.

# 4 Sample Running Time:

$ time scala project1.scala 1000000 4

real 0m3.716s

user 0m7.367s

sys 0m0.643s

the ration of CPU time and REAL time is 2.15

# 5 Capability

In this program, BigInt is used to handle possible extreme large inputs as numbers larger than the maximum of type of Long. However, there is no function for calculate a square root of a BigInt, thus I use Heron's methods to test whether a BigInt a perfect square number. This program could handle extreme large number but takes long time. However, type of BigInt causes overall low efficiency when inputs are not large enough comparing with usage of type of Long.

$ time scala project1.scala 100000000 24

1

9

20

...

...

342988229

518925672

816241996

real 19m30.425s

user 50m0.483s

sys 0m31.973s

```
//User Heron's method to determine maximum root square not exceeds n
def isSqrt(n: BigInt, root: BigInt): Boolean = {
val lowerBound : BigInt = root.pow(2)
val upperBound : BigInt = (root + 1).pow(2)
return lowerBound.compare(n) <= 0 && n.compare(upperBound) < 0 }

def sqrt(n: BigInt) : BigInt = {
var bitLength = n.bitLength
var root = BigInt.apply(1) « (bitLength/2)
while (!isSqrt(n, root)) {root = (root + n / root) / 2}
return root
}
```

# 6 Bonus Parts

To enable Akka remoting, akka need to be configured. As my running environment is UF HPC (http://www.hpc.ufl.edu), I cannot control running nodes so they must be dynamically configured. Here is the code:

```
val hostname = InetAddress.getLocalHost.getHostName
val config = ConfigFactory.parseString(
(s""" akka {
actor {provider = "akka.remote.RemoteActorRefProvider" }
netty.tcp { hostname = "$hostname" port = 2552 }
}
}""")
```

So every remote machine will automatically get configured to listen on its hostname. To manage 10 remote machine, I added a supervisor to manager all of them. On each node, a boss actor and several workers actors will perform local computation. Local bosses use remote lookup to locate super boss and request a large range of number from it. In fact there is no much difference between super boss and local bosses. I didn't include failure detector in this project but if one node failed, another one who first complete its job will take this task until all tasks are reported as completed.

To run on HPC, additional scripts should be made to queue and use MPI to start up 10 instance on different 10 machines. Each boss actor request 1,000,000 per time from the supervisor and supervisor counts how many boss are running. After all numbers are assigned and running boss actors decreased to 1, it shutdown the actor system. Performance of hard disk I/O on HPC is bad when compiling or operating small files which may cause exceptions so all scripts must be compiled before being executed.

```
$mpirun -np 10 -ppn 1 java -classpath $CLASSPATH project1
```

Demo are made to calculate 24 continous integers from 1 to 10,000,000,000. It tooks 1:19:00 to finish the computation and here is the result. Note there are ten "Client Started" messages indicates that there are ten boss actors started on ten machines. Maximum solution is 8,079,962,876. (I put them into one line to save space)

```
Client Started Client Started Client Started Client Started 1 9 20 25 44 76 121 197
304 353 540 856 1301 2053 3112 3597 5448 8576 12981 20425 128601 30908 Client
Started 35709 Client Started 54032 Client Started 84996 202289 Client Started 306060
Client Started 353585 Client Started 534964 841476 1273121 2002557 3029784 3500233
5295700 8329856 12602701 19823373 29991872 34648837 52422128 82457176 124753981
196231265 296889028 342988229 8079962876 518925672 5136834684 1234937201 816241996
3395233545 1942489369 2938898500
```