Exercise 3.1

Question 1

Benchmark.java file has been created to implement this exercise.

All the outliers are closed. Only the notepad++ and Adobe Acrobat Reader and command prompt are running.

-System Info

```
# OS:   Windows 10; 10.0; amd64
# JVM:  Oracle Corporation; 1.8.0_181
# CPU:  Intel64 Family 6 Model 142 Stepping 10, GenuineIntel; 8 "cores"
# Date: 2018-09-19T23:11:44+0200
```

Processor : i5 , 1.8 GHz, 8 GB RAM

-Mark1

Running Mark1 several times on count = 1 million

5.1 ns
4.6 ns
4.6 ns

Running Mark1 several times on count = 100 million

0.0 ns
0.0 ns
0.0 ns

The results are almost identica with Microbenchmarks.

-Mark2

The results from Mark 2 after running several times.

24.7 ns
24.7 ns
24.7 ns

The results are almost identica with Microbenchmarks.

-Mark3

Results from Mark 3
24.7 ns
24.8 ns
24.7 ns
24.8 ns
24.7 ns
24.7 ns
24.7 ns
24.7 ns
24.8 ns
24.9 ns

The results are almost identica with Microbenchmarks.

-Mark4

24.7 ns +/- 0.020

Not any suprises.

-Mark5

Results from Mark5

```
284.5 ns +/-   709.51          2
156.5 ns +/-    44.99          4
128.0 ns +/-   115.14          8
117.3 ns +/-   122.09         16
 35.6 ns +/-     0.02         32
 34.7 ns +/-    19.41         64
 37.8 ns +/-    29.05        128
 29.3 ns +/-     4.03        256
 28.8 ns +/-     1.92        512
 28.4 ns +/-     1.14       1024
 28.4 ns +/-     0.94       2048
 28.0 ns +/-     0.34       4096
 26.3 ns +/-     0.75       8192
 26.3 ns +/-     1.29      16384
```

```
26.4 ns +/-    0.69      32768
24.7 ns +/-    0.08      65536
24.7 ns +/-    0.07     131072
24.7 ns +/-    0.03     262144
24.7 ns +/-    0.02     524288
24.7 ns +/-    0.03    1048576
25.3 ns +/-    1.30    2097152
24.7 ns +/-    0.01    4194304
24.9 ns +/-    0.16    8388608
24.8 ns +/-    0.12   16777216
```

-Mark6

```
multiply                     512.0   1028.26          2
multiply                     170.7     89.90          4
multiply                     170.7    171.59          8
multiply                     124.5     45.13         16
multiply                     142.2     89.87         32
multiply                      54.2     31.76         64
multiply                      53.8     50.04        128
multiply                      39.3     10.63        256
multiply                      57.7     31.10        512
multiply                      35.5      4.67       1024
multiply                      32.9      0.62       2048
multiply                      40.0     17.06       4096
multiply                      26.4      0.81       8192
multiply                      26.4      0.47      16384
multiply                      25.0      0.79      32768
multiply                      26.2      1.68      65536
multiply                      24.7      0.09     131072
multiply                      33.9     14.44     262144
multiply                      24.7      0.02     524288
multiply                      24.7      0.02    1048576
multiply                      26.0      3.36    2097152
multiply                      25.0      0.38    4194304
multiply                      25.0      0.62    8388608
multiply                      25.2      0.83   16777216
```

Question 2

-System Info

```
# OS:   Windows 10; 10.0; amd64
# JVM:  Oracle Corporation; 1.8.0_181
# CPU:  Intel64 Family 6 Model 142 Stepping 10, GenuineIntel; 8 "cores"
# Date: 2018-09-19T23:11:44+0200
```

Processor : i5 , 1.8 GHz, 8 GB RAM

-Mark7

The result of different functions

```
pow                          72.6      0.09    4194304
exp                          53.5      0.15    8388608
log                         22.7       0.02   16777216
sin                         48.0       0.15    8388608
cos                         103.7       2.40    4194304
tan                          97.3       0.49    4194304
asin                        350.9      91.20    2097152
acos                         178.0       1.37    2097152
atan                         38.0       0.05    8388608
```

It can be seen from the above result asin has very high sd which means there was some backgroud task running,
it might be garbage collector or some other external disturbance.

The output is similar to Microbenchmarks.

Exercise 3.2

1. There are several fluctution in sd sometimes it is more and sometimes less. In my POV, this is caused by the external disturbances.

2.

```
# OS:   Windows 10; 10.0; amd64
# JVM:  Oracle Corporation; 1.8.0_181
# CPU:  Intel64 Family 6 Model 142 Stepping 10, GenuineIntel; 8 "cores"
# Date: 2018-09-20T00:30:52+0200
```

-Run 1

```
hashCode()                    2.7 ns       0.00  134217728
Point creation               44.6 ns       1.72    8388608
Thread's work              5325.2 ns       2.96      65536
Thread create              1042.4 ns     386.19     262144
Thread create start      164548.3 ns    5879.01       2048
Thread create start join   127590.0 ns    402.70       2048
ai value = 1392580000
Uncontended lock               10.1 ns       0.07   33554432
```

-Run 2

```
hashCode()                    2.7 ns       0.03  134217728
Point creation               50.2 ns      15.57    8388608
Thread's work              5326.0 ns       2.89      65536
Thread create              1019.4 ns     360.39     524288
Thread create start       85387.7 ns   28241.12       4096
Thread create start join   300106.9 ns   4540.61       1024
ai value = 1413060000
```

In my point of view, "The thread create", "thread create start" and "thread create start join" should have the mean in an non decreasing order as their are more operation included prior to the previous one, however, in run1 "thread create start" is higher than "thread create start join" but the result should be as of run2. Because of these uncertainty it is really difficult to do benchmarking which is also mentioned in microbenchmarking.

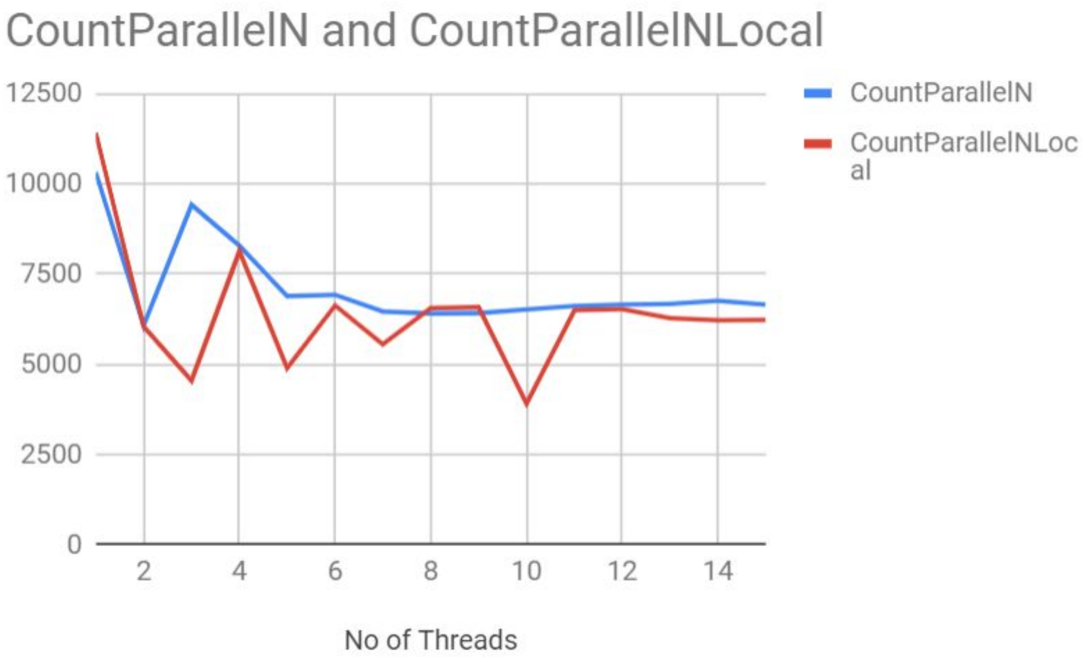
Exercise 3.3

1. I only measure from 1 to 15 threads as it takes too long to operate.

```
# OS:   Windows 10; 10.0; amd64
# JVM:  Oracle Corporation; 1.8.0_181
# CPU:  Intel64 Family 6 Model 142 Stepping 10, GenuineIntel; 8 "cores"
# Date: 2018-09-20T20:52:36+0200
countSequential                9343.1 us      29.51         32
countParallelN          1          10310.1 us     923.58         32
countParallelNLocal     1        11404.8 us    1541.81         32
countParallelN          2           6073.1 us      35.77         64
countParallelNLocal     2         6037.3 us     139.64         64
countParallelN          3           9413.1 us     164.41         32
countParallelNLocal     3         4543.8 us      45.23         64
countParallelN          4           8280.1 us     828.03         64
countParallelNLocal     4         8134.2 us     201.06         32
countParallelN          5           6884.4 us    1768.99        128
```

countParallelNLocal	5	4891.7 us	1910.51	128
countParallelN	6	6922.4 us	29.28	64
countParallelNLocal	6	6625.2 us	51.47	64
countParallelN	7	6460.5 us	63.88	64
countParallelNLocal	7	5550.5 us	1371.92	128
countParallelN	8	6405.0 us	319.63	64
countParallelNLocal	8	6549.9 us	208.47	64
countParallelN	9	6413.9 us	192.77	64
countParallelNLocal	9	6576.1 us	348.44	64
countParallelN	10	6514.2 us	129.52	64
countParallelNLocal	10	3912.6 us	1462.95	128
countParallelN	11	6609.1 us	87.54	64
countParallelNLocal	11	6493.8 us	56.73	64
countParallelN	12	6652.9 us	42.49	64
countParallelNLocal	12	6525.8 us	195.74	64
countParallelN	13	6672.7 us	60.89	64
countParallelNLocal	13	6274.7 us	48.78	64
countParallelN	14	6756.5 us	255.92	64
countParallelNLocal	14	6213.0 us	55.88	64
countParallelN	15	6651.0 us	65.96	64
countParallelNLocal	15	6223.2 us	75.98	64

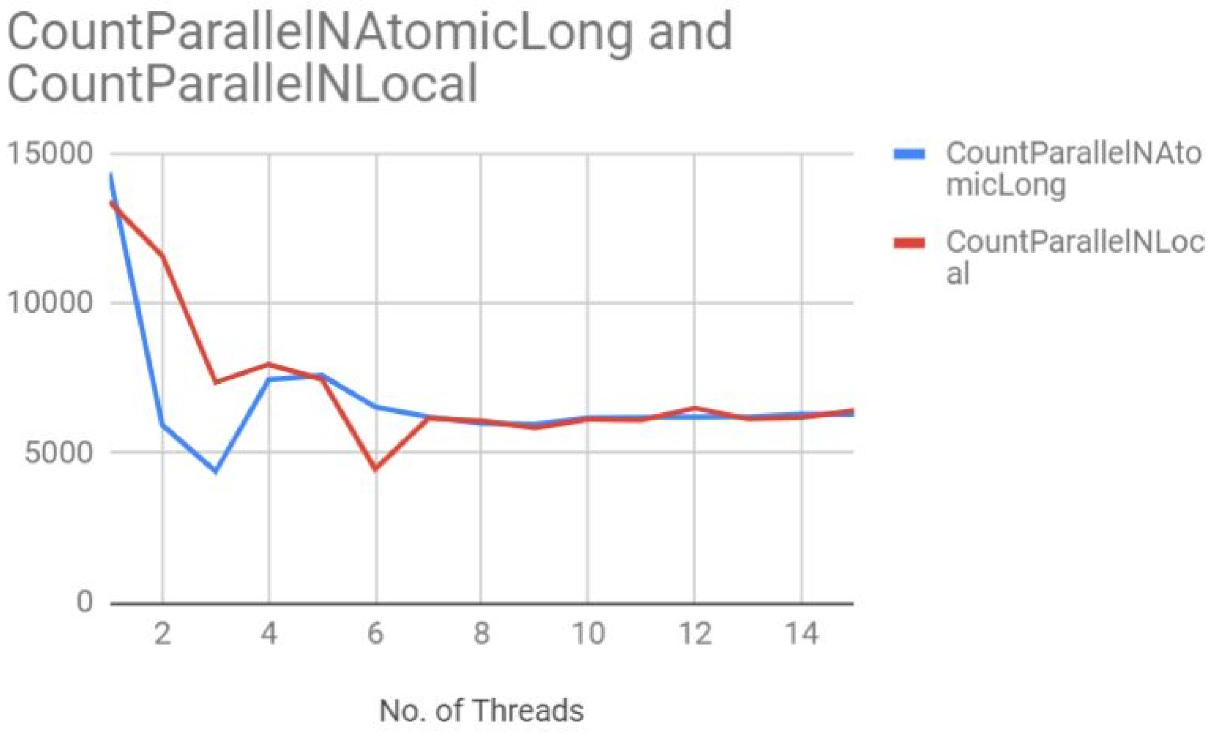
2. Graph done in google sheets



3. It can be seen from the graph that the performance was poor in the beginning with 1 thread and the performance was good with 10 threads. As the system has 8 cores in it, it is acceptable that it has given best performance with 10 threads. But the surprising thing is that, the performance with 2 threads was close with the 10 threads. We can see the increase in the threads did not scale the performance which proves that many threads does not increase the performance, however, the number of threads should be used as per the cores of the system.

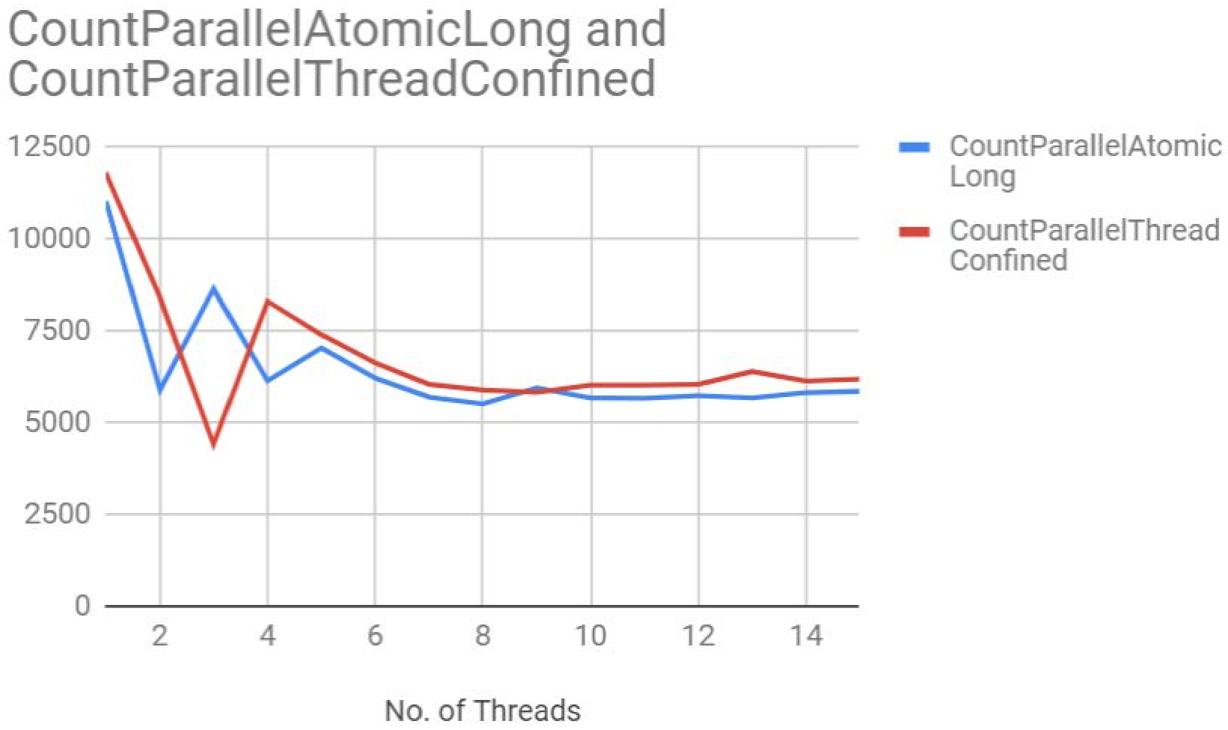
Also CountParallelNLocal performs better than that of CountParalleN

4. Graph



From the graph, I can say that the performance of AtomicLong is somehow better. Its performance is good than local except at thread 6 which is obvious due to some external disturbances. Also, it might perform equal at some case. However, I prefer to use the built in classes rather than making it from scratch as I don’t have to worry about the thread safe issues.

5.



The benchmarking shows that both the methods are giving somehow identical performance but if we be precise thread confined operation is bit slower.

Exercise 3.4

The solution is implemented in TestCache.java and changes has been done regarding the amount of work.

1. 1960612.5 ns 643880.11
2. 1399833.6 ns 417039.84
3. 983824.3 ns 357847.36
4. 911967.2 us 312246.91
5. 930304.5 us 348268.94
6. 980594.3 us 370899.42
7. As discussed in the lecture, Memoizer1 and Memoize2 performs worst. Memoizer4 is giving the best result. I believe that Memoizer5 is performing slower than Memoizer3 and Memoizer4 becuase of some curse from external outliers or due to some disturbance from garbage collector and others.
8. If I have time then I would like to experiment it with higher amount of work and may be doing the experiment in linux instead of Windows as there are many background tasks running in Windows.