Practical Concurrent and Parallel Programming IV

Jørgen Staunstrup

IT University of Copenhagen

Friday 2020-09-18

Starts at 8:00

Plan for today

- Performance measurements: motivation and introduction
- Calculating means and variance (pitfalls)
- Measurements of thread and lock overhead
- Questions from you (cost of volatile? and cost of changing thread?)
- Algorithms for parallel computing

About Jørgen

Datalog/Computer scientist Aarhus University: 1975

Taught first course on Concurrency: 1978 (USC, Los Angeles)
It was an on-line course !!

Joined ITU in 2001, retired in 2014.

Teaching MSc course Mobile App Development at ITU since 2016

Hand-out material

- Peter Sestoft: Microbenchmarks in Java and C sharp
- CodeForBenchmarkNote/Benchmark.Java
- Slides from lecture
- Exercises week 4
- Recording of lecture (uploaded after the class on 09-18)

Slides from lecture on Computer Arithmetic:
https://www.itu.dk/people/sestoft/bachelor/computernumbers.pdf

Take a look at the last page with references (it is a goldmine!!)

Motivation (performance measurements)

How expensive ?

~600 ns to create (on this laptop)

~20 times more time than creating a simple object

40000 ns to start a thread !!!

Today: How to get such numbers !

(Performance) Measurements

- Key in many sciences (experiments, observations, predictions, ...)
- A bit of statistics
- A bit of numerical analysis
- A bit of computer architecture (number representation)
- Code for measuring execution time

Based on Microbenchmarks in Java and C# by Peter Sestoft (see benchmarkingNotes.pdf in material for this week)

All numbers in these slides were measured in September 2020 on a:

Intel(R) Core(TM) i5-1035G4 CPU @ 1.10GHz, 1498 Mhz, 4 Core(s), 8 Logical Processor(s)

Example multiplication

A simple Timer class for Java

Works on all platforms (Linux, MacOS, Windows)

[Code for the benchmark note and exercises]

Mark2 in BenchMark.java

Let us try it.

Automating multiple samples (Mark3)

23.9 ns
23.9 ns
23.7 ns
23.9 ns
23.8 ns
23.7 ns
23.7 ns
23.7 ns
23.7 ns
23.7 ns

What is the running time?

What should you report as the result, when the observations are:

30.7 ns 30.3 ns 30.1 ns 30.7 ns 30.5 ns 30.4 ns 30.9 ns 30.3 ns 30.5 ns 30.8 ns ??

Mean: 30.4 ns

What if they are:

30.7 ns 100.2 ns 30.1 ns 30.7 ns 20.2 ns 30.4 ns 2.0 ns 30.3 ns 30.5 ns 5.4 ns ??

Mean: 31.0 ns

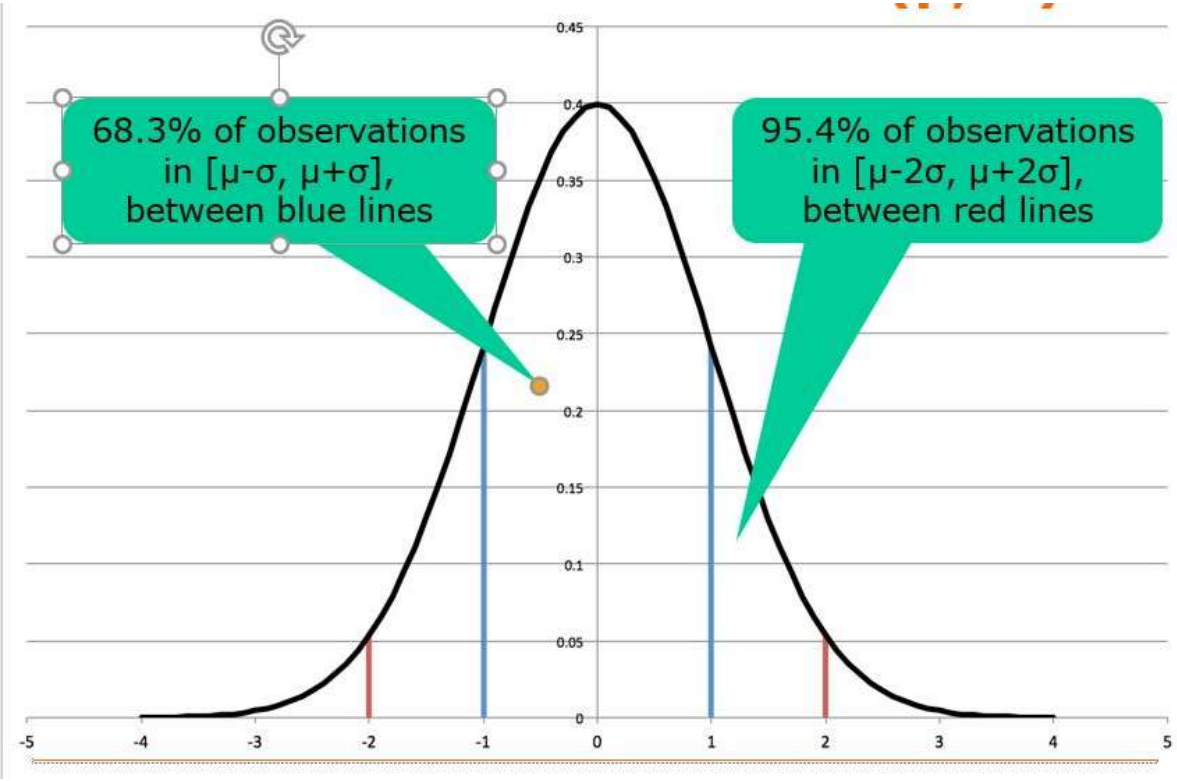
and variance

Variance

$$\sigma^2 = \frac{1}{n(n-1)} \left(n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2 \right)$$

see also https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance

Normal Distribution



What is the running time?

What should you report as the result, when the observations are:

30.7 ns 30.3 ns 30.1 ns 30.7 ns 50.2 ns 30.4 ns 30.9 ns 30.3 ns 30.5 ns 30.8 ns ??

Mean: 32.5 ns Standard deviation: 6.2

50.2 is an outlier

because there is a probability of less than 4.6 % that 50.2 is a correct observation

Warning

$$\sigma^2 = \frac{1}{n(n-1)} \left(n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2 \right)$$

Beware:

Floating-point numbers

IEEE 754 binary32 and binary64
Which you know as and in Java (and C#)

IEEE 754 decimal128
Java's

The number 1.0527 can be witten as: 10527 / 10 ^ 4 (fraction)

representation is 10527 4

But not all numbers can be represesented precisely in a finite numer of bits !!!
1/10 = 0.0001100110011001100110011…

See Wikipedia page on accuracy problems

So we cannot represent 0.10 Kr. or $0.10 exactly

Nor 0.01 Kr. or $0.01 exactly

Digit loss

Beware of cancellation when subtracting numbers that are close to each other:

Therefore:

```
```

can be problematic.

How to do it:
https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance

Benchmark code (Mark6)

see Benchmark.java in code for this week

Cost of Threads

A thread is an object,so let us start finding the cost of creating a simple object.

Benchmark code:

So object creation is: ~ 31 ns

Thread creation

Takes 600 ns ~ 20 times slower than Point creation

Thread create + start

Takes ~ 40000 ns

- So a lot of work goes into setting up a thread
- Even after creating it
- Note: does not include executing the loop (why?)

See more examples in BenchMark.java

Question from week 3

Also tried the same with and an extra Thread t2 to see if was optimized away. Apparently, it was not.

Question from LearnIT

Cost of volatile? Link

Exercise 4.6

Algorithms for parallel computing

Quicksort

Problem: The first iteration through ALL the data (O(n)) only activates one thread,

while the rest are waiting.

The second iteration only activates two threads etc.

The resulting speed-up using 8 cores has been reported as 3.6.

CountPrimes

Problem: Computing the prime factors of small numbers much faster than for large numbers.

Work balancing is IMPORTANT.

Problem-heap