Doing experiments in practice: measurements and statistics

Sid Touati
Professor in computer science, university Côte d'Azur

Requirements

In order to be able to do this lab, you need to install the following software on your test machine:

1. Linux operating system (64bits).
2. C compiler: `gcc` for instance.
3. `libgomp` library (it is OpenMP library, which is normally installed by default if you install gcc, but check it anyway)
4. R statistical software: `https://www.r-project.org`
5. Rstudio (optional, free version): `https://rstudio.com/products/rstudio/download`

1 Introduction: what will we do during this lab ?

This lab gives you an example of experiments in high performance computing, and a first experience in analysing data with rigorous statistics. The purpose is to compare between two code versions, and select the fastest one. We take as example the matric-matrix multiplication benchmark, which is known as a massively parallel code. The C-code is given in the package.

This code is implemented with OpenMP parallelism, so its execution will generate multiple threads. The number of OpenMP threads in our experiments must not exceed the number of physical cores on your machines. So please check the number of cores of your computer before starting your experiments. You can check the number of cores on your computer by searching in the file content named `/proc/cpuinfo`.

To fix the idea, the purpose of this lab is to check if an execution with 4 threads for instance is faster or not than an execution with 3 threads on your machine.

2 Example of measurements and collecting data: program execution times

The benchmark that we want to execute and measure is the C code called `matrixmatrixmultiply.c`. It is stored in the directory named `benchmark`.

2.1 Compiling the C benchmark

In order to compile the C code, use the following command on the Linux shell:

```
gcc -O2  -fopenmp ./matrixmatrixmultiply.c -o ./matrix
```

The above command creates an executable binary called `matrix`. This executable uses OpenMP library (thread parallelism).

2.2 Executing the C benchmark with different threads numbers

In order to execute the `matrix` executable code, we need to specify the total number of threads to use during the execution. For instance, if we want to run the program with 3 threads, we use the following command on the Linux shell:

```
 export OMP_NUM_THREADS=3
 ./matrix
```

In order to measure the execution time of the above program, we use the command named `time` as follows:

```
 export OMP_NUM_THREADS=3
time ./matrix
```

The `time` command reports many timing measures, which are the real execution time, the user time and the system time. In this lab, we will focus on the real time only. We use the following command line options in order to capture only this measure into a file named `toto` for instance.

```
 export OMP_NUM_THREADS=3
/usr/bin/time -f "%e" -o toto  ./matrix
```

You can check here that the file `toto` contains the measured execution time.

2.3 Repeating the experiments and collecting performance data

Running a program only once is not sufficient to make confident statistics, especially if the execution times vary. So we will have to repeat the experiments (running of a program) a large number of times, say 50 times. The collected running times must be saved into a file to be analysed later. We can use a shell command to repeat the execution of the benchmark, and save the measured running time into a file. For instance, `bench1.txt` saves the running times of the benchmark run with 3 threads, and `bench2.txt` saves the running times of the benchmark run with 4 threads, 50 times each. The following commands repeat the execution of the benchmark with 3 threads, 50 times, measures its real execution time and save it into the file named `bench1.txt` .

```
export OMP_NUM_THREADS=3
for i in ‘seq 1 50‘ ; do /usr/bin/time -f "%e" -o toto ./matrix ; cat toto >> bench1.txt ; done
```

Same for repeating the benchmark with 4 threads and saving the running times into the file `bench2.txt`.

```
export OMP_NUM_THREADS=4
for i in ‘seq 1 50‘ ; do /usr/bin/time -f "%e" -o toto ./matrix ; cat toto >> bench2.txt ; done
```

From now, we produced two files `bench1.txt` and `bench2.txt` that contains 50 numbers each. The next section explains how to analyse these data.

2.4 If you want to skip this section

If you did not succeed in this section, you can skip it and retrieve some sample of data in the directory named `Data`. You will find two files `bench1.txt` and `bench2.txt` containing the collected execution times of two programs respectively. You can use these two files in the following sections devoted to statistics.

3 Analysing data with R

In this section, you will learn how to visualise and analyse data with `R`. After launching the graphical interface of `R` (`R-Studio` for instance), start by loading some libraries by typing the following commands in the R console (install these libraries with R if requested):

```
library(stats)
library(graphics)
library(vioplot)
```

Now, you can load your data stored into the two files `bench1.txt` and `bench2.txt` by typing the following commands in the R console (modify fie paths and names if needed):

```
X <- read.csv("bench1.txt", header=F)$V1
Y <- read.csv("bench2.txt", header=F)$V1
```

Now, the two sets of data `X` and `Y` contain the collected execution times of two programs respectively. If you want to compute initial empirical statistics of your data, type the following commands in the R console:

```
summary(X)
summary(Y)
```

3.1 Boxplots

If you want to visualise your data graphically, you can use boxplots as follows:

```
boxplot(X,Y,names=c("bench1","bench2"),col=c("blue","pink"),ylab="seconds", main="Execution times")
```

Boxplots allow to quickly visualise maximum, minimum, quartiles and median. What do you conclude from the graphical visualisation of the data ? Is `X` lower than `Y` ? or `Y` lower than `X` ?

3.2 Violinplots

Another graphical visualisation is violinplots. They are like boxplots, but the shape of the figure follows the density function of the sample data. You can draw violin plots by typing the following commands in the R console:

```
vioplot(X,Y, names=c("bench1", "bench2"), ylab="seconds", main="Execution times", col=c("blue","pink"))
```

What do you conclude from the graphical visualisation of the data ? Is `X` lower than `Y` ? or `Y` lower than `X` ?

3.3 Histograms

In order to have a quick look on the distribution of your data, you can visualise histograms as follows:

```
hist(X, main="Distribution of execution times")
hist(Y, main="Distribution of execution times")
```

With histograms, the X-axis corresponds to the distinct data values, and the Y-axis correspond to the frequency of these data in the sample. What do you conclude from the graphical visualisation of the data ? Do you observe a central value (a mean)? What would be the metric of comparison of interest in your case, mean or median ?

Now, assume that we repeat the experiences again and again, are you sure that you will get the same conclusions based on the graphical visualisation ? The next section will show how we can do statistical testing based on samples of data to make conclusions with a high degree of confidence.

4 Statistical testing with R

4.1 Comparing between two theoretical means

If the sizes of `X` and `Y` are big enough, you can use directly the student t-test[1]. Otherwise, you have first to check if if `X` and `Y` follow normal (gaussian) distribution as explained below.

4.1.1 Testing if a data sample follows a normal (gaussian) distribution

The following R command performs the normality test using Shapiro-Wilk method:

```
shapiro.test(X)
```

The null hypothesis of the test is: "the sample data `X` follows a normal distribution". The test comptes a *p*-value, which is the risk (probability) of doing an error if you reject the null hypothesis. If *p*-value is high (typically higher than 5%), than we must not reject the null hypothesis, thus we conclude that the sample data `X` follow a normal distribution. If *p*-value is low (typically lower than 5%), than we can reject the null hypothesis and conclude that the sample data `X` does not follow a normal distribution. Do the same test for the data sample `Y`.

4.1.2 Using the student t-test

If the data sample does not follow normal distribution, its size must be large enough, otherwise the risk level computed by the student t-test would be incorrect. The student t-test allows to compare between two theoretical means of two random variables `X` and `Y`. Let us denote by μ_X, μ_Y the two theoretical means of `X` and `Y`. Note that the exact values of μ_X, μ_Y cannot be computed, we can only compute their confidence intervals.

Testing if $\mu_X \leq \mu_Y$: The following R command performs this statistical test:

```
t.test(X, Y, "greater")
```

The null hypothesis of the above test is : "$\mu_X > \mu_Y$". The test computes a *p*-value, which is the risk level of rejecting the null hypothesis. If the *p*-value is low enough (for instance lower than 5%), we can reject the null hypothesis and conclude that "$\mu_X \leq \mu_Y$".

Testing if $\mu_X \geq \mu_Y$: The following R command performs this statistical test:

```
t.test(X, Y, "less")
```

The null hypothesis of the above test is : "$\mu_X < \mu_Y$". The test computes a *p*-value, which is the risk level of rejecting the null hypothesis. If the *p*-value is low enough (for instance lower than 5%), we can reject the null hypothesis and conclude that "$\mu_X \geq \mu_Y$".

Testing if $\mu_X \neq \mu_Y$: The following R command performs this statistical test:

```
t.test(X, Y, "two.sided")
```

The null hypothesis of the above test is : "$\mu_X = \mu_Y$". The test computes a *p*-value, which is the risk level of rejecting the null hypothesis. If the *p*-value is low enough (for instance lower than 5%), we can reject the null hypothesis and conclude that "$\mu_X \neq \mu_Y$".

[1]The commands `length(X)` and `length(Y)` compute the sizes of `X` and `Y`.

4.2 Comparing between two theoretical medians

If the sizes of `X` and `Y` are big enough, you can use directly the Wilcoxon-Mann-Whitney's test. Otherwise, you have first to check if if `X` and `Y` follow the location shift model as explained below.

4.2.1 Testing if `X` and `Y` follow the location shift model

Kolmogorov-Smirnov test allows to make such verification by using the following command:

```
ks.test(X-median(X),Y-median(Y))
```

The null hypothesis of this test is: "$X-\text{median}(X)$ and $Y-\text{median}(Y)$" are issued from the same distribution. It computes a p-value. If this p-value is low enough (for instance lower than 5%), we can reject that `X` and `Y` follow the location shift model. Otherwise, if p-value is high enough, we accept that `X` and `Y` follow the location shift model, and hence we can safely use the Wilcoxon-Mann-Whitney's test below. Note that sometimes, this test fails if the sample data `X` or `Y` contain some ex-aequos values[2].

4.2.2 Using Wilcoxon-Mann-Whitney's test

Testing if median$(X) \leq$ median(Y) The following R command performs this statistical test:

```
wilcox.test(X, Y, "less")
```

The null hypothesis of the above test is : "$F(X) < F(Y)$" (comparing cumulative distribution functions). The test computes a p-value, which is the risk level of rejecting the null hypothesis. If the p-value is low enough (for instance lower than 5%), we can reject the null hypothesis and conclude that "$\text{median}(X) \leq \text{median}(Y)$".

Testing if median$(X) \geq$ median(Y) The following R command performs this statistical test:

```
wilcox.test(X, Y, "greater")
```

The null hypothesis of the above test is : "$F(X) > F(Y)$" (comparing cumulative distribution functions). The test computes a p-value, which is the risk level of rejecting the null hypothesis. If the p-value is low enough (for instance lower than 5%), we can reject the null hypothesis and conclude that "$\text{median}(X) \geq \text{median}(Y)$".

Testing if median$(X) \neq$ median(Y) The following R command performs this statistical test:

```
wilcox.test(X, Y, "two.sided")
```

The null hypothesis of the above test is : "$F(X) \neq F(Y)$". The test computes a p-value, which is the risk level of rejecting the null hypothesis. If the p-value is low enough (for instance lower than 5%), we can reject the null hypothesis and conclude that "$\text{median}(X) \neq \text{median}(Y)$".

4.2.3 Remark

Statistical tests do not provide certitudes, they are good tools that help you to make conclusions based on rigorous statistics. Sometimes, it may happen that the student t-test does not allow to clearly compare between μ_X and μ_Y. That is, we may not conclude that $\mu_X \neq \mu_Y$, nor $\mu_X \leq \mu_Y$ nor $\mu_X \geq \mu_Y$. The same situation may hold for Wilcoxon-Mann-Whitney's test when comparing between medians.

[2] In the continuous model of probability theory, the probability of observing exactly the same values is zero.

5 Writing a short report of your experiments and data analysis

Write a report using LATEXthat describe what you did during this lab. It must contain the following sections:

- Hardware setup: explain the hardware configuration of your test machine: processor model, processor architecture, CPU frequency, caches sizes, memory size, etc.
- Software setup: which compiler and version did you use, which OpenMP version.
- Experimental setup: describe how did you collect your performance data, number of repetitions, what was the general environment of your system, etc.
- Data visualisation: put the boxplots, violinplots and histograms.
- Statistical tests using the data: explains all the statistical tests that you did, and report your conclusions based on theses statistical tests: what is the faster code version in average ? what is the faster code version in median ? etc.