Introduction

In this report, we compare the performance, in term of *run time*, of the *parallel quicksort*, as implemented by **Arnaud Legrand** (https://github.com/alegrand/M2R-ParallelQuicksort)

Experiment setup

The tests where conduced on a 2011 MacBook Pro, with an `Intel(R) Core(TM) i7-2635QM CPU @ 2.00GHz` 4 core processor (with intel's hyperthreading technologies this means 8 virtual cores), 8 Gigabytes of ram, and running Fedora Linux 35.

Fixing problems with the original experiment

Increasing the robustness of the testing script

A test script is provided by default on this project :

```
cat scripts/run_benchmarking.sh
```

```
OUTPUT_DIRECTORY=data/‘hostname‘_‘date +%F‘
mkdir -p $OUTPUT_DIRECTORY
OUTPUT_FILE=$OUTPUT_DIRECTORY/measurements_‘date +%R‘.txt

touch $OUTPUT_FILE
for i in $(seq 1000000 1000000 10000000) ; do
    for rep in ‘seq 1 5‘; do
        echo "Size: $i" >> $OUTPUT_FILE;
        ./src/parallelQuicksort $i >> $OUTPUT_FILE;
    done ;
done
```

While this script is fine, it can be enhanced to be made more robust. As suggested by Arnaud Legrand, we will now make the script in Python, and the main modification will be the interleaving of the execution. Aditionnaly, we will collect data in static increment instead of powers of 10 as done originally :

```
cat scripts/run_benchmarking.sh # TODO
```

```
OUTPUT_DIRECTORY=data/‘hostname‘_‘date +%F‘
mkdir -p $OUTPUT_DIRECTORY
OUTPUT_FILE=$OUTPUT_DIRECTORY/measurements_‘date +%R‘.txt

touch $OUTPUT_FILE
for i in $(seq 1000000 1000000 10000000) ; do
    for rep in ‘seq 1 5‘; do
        echo "Size: $i" >> $OUTPUT_FILE;
        ./src/parallelQuicksort $i >> $OUTPUT_FILE;
    done ;
done
```

Modifying the `MakeFile` for optimisation

As pointed out by the professor **Arnaud Legrand** again the executable is compiled with `-O0`. For a performance test, it seems more relevant to use `-O3`, even if `-O0` is the default :

Results

We have 2 objectives in this experiment: -Finding if the parallel version is faster -If yes, finding at what input size it is faster, because of course, the overhead of parallelisation is never worth the cost on very small inputs, so there will be a crosspoint when parallel becomes faster

```
cat src/Makefile
```

```
parallelQuicksort: parallelQuicksort.o

PEDANTIC_PARANOID_FREAK =       -g -Wall -Wshadow -Wcast-align \
                -Waggregate-return -Wmissing-prototypes -Wmissing-declarations \
                -Wstrict-prototypes -Wmissing-prototypes -Wmissing-declarations \
                -Wmissing-noreturn  \
                -Wpointer-arith -Wwrite-strings -finline-functions -O3

REASONABLY_CAREFUL_DUDE =   -g -Wall -O2
NO_PRAYER_FOR_THE_WICKED =  -w -O3
WARNINGS =          $(PEDANTIC_PARANOID_FREAK)
CFLAGS = $(WARNINGS) -pthread -lrt -std=c99
LIBS =

%: %.o
    $(CC) $(INCLUDES) $(DEFS) $(CFLAGS) $^ $(LIBS) -o $@

%.o: %.c
    $(CC) $(INCLUDES) $(DEFS) $(CFLAGS) -c -o $@ $<

clean:
    rm -f parallelQuicksort *.o *~
```

```
library(dplyr)
```

```
##
## Attachement du package : 'dplyr'
```

```
## Les objets suivants sont masqués depuis 'package:stats':
##
##     filter, lag
```

```
## Les objets suivants sont masqués depuis 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
require(scales)
```

```
## Le chargement a nécessité le package : scales
```

```
library(vroom)
nouveau_resultat= read.csv("data/eduroam-109015.grenet.fr_2021-11-18/measurements_14:44.csv")

base_path = "data/eduroam-109015.grenet.fr_2021-11-18/"
files = paste(base_path, list.files(base_path), sep = "")
df <- vroom(files[grep(".csv", x=files)])
```

```
## Rows: 525 Columns: 3
```

```
## -- Column specification --------------------------------------------------------
```

```
## Delimiter: ","
## chr (1): Type
## dbl (2): Size, Time
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

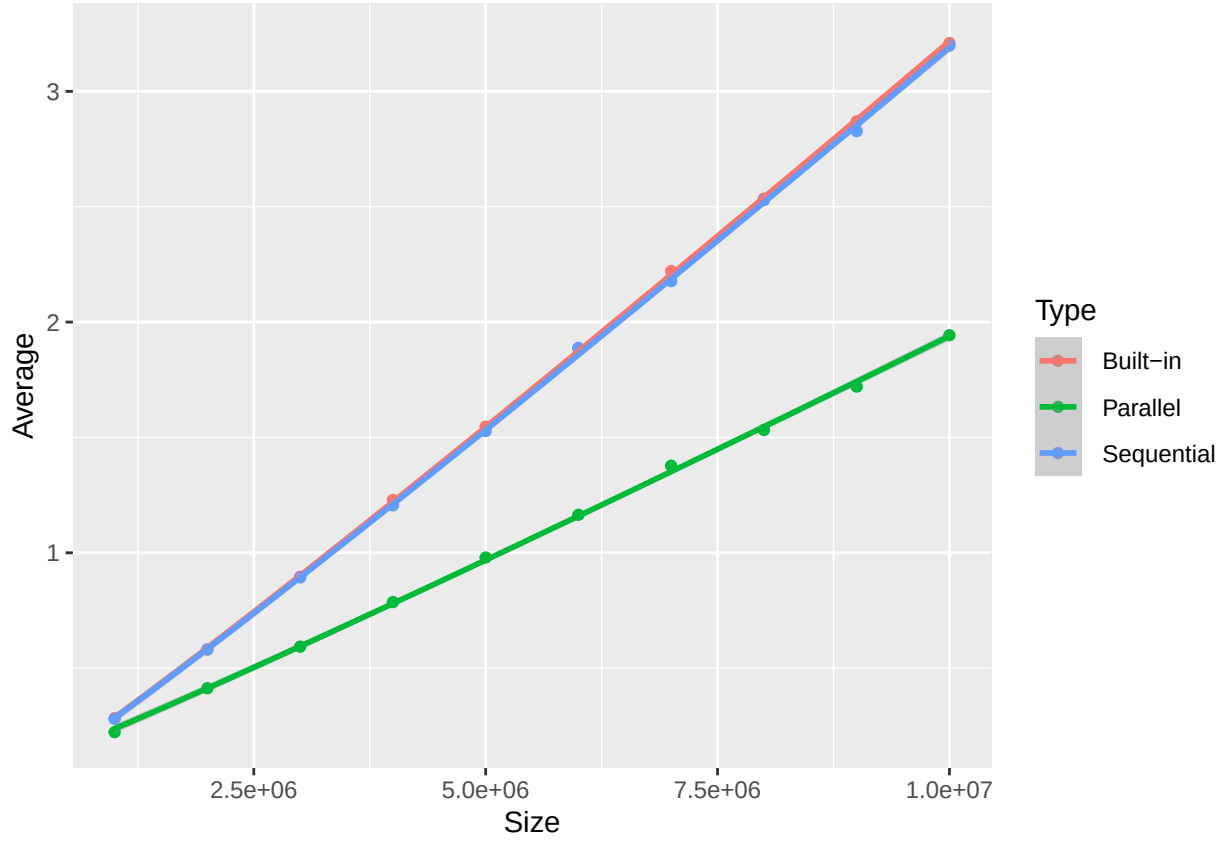
```
df = read.csv("data/eduroam-109015.grenet.fr_2021-11-18/measurements_16:50.csv")


df = df %>% group_by(Type, Size) %>% summarise(Average=mean(Time))
```

```
## `summarise()` has grouped output by 'Type'. You can override using the `.groups` argument.
```

```
p <- ggplot(df, aes(x=Size, y=Average, color=Type))
p <- p  + geom_point()  + geom_smooth(method = 'lm',formula = y ~I(x*log(x)))

print(p)
```




```
#p <- p + scale_x_log10(breaks = trans_breaks("log10", function(x) 10^x))+scale_y_log10(breaks = trans_
#p <- p + stat_function(fun= function(x) x * log10(x))
```

Avec la dichotomie nous trouvons expérimentalement, sur mon ordinateur, que l'algorithme parallèle deviens plus rapide autour de 5×10^5