Analysis of Sequential vs Multi-Threaded (Parallel) QuickSort Algorithm

Olaniyan Folajmi & Gabriele Degola

November 2021

Parallel Quicksort Analysis

In this work, we will like to compare the performance of the sequential quick sort algorithm to its sequential counterpart. While the parallel version of the quicksort algorithm is slower for smaller input sizes, for sufficiently large input sizes, parallel quicksort runs faster than the sequential version.

Extensions to base code

Our experiments are based on the initial code available here. However, we made a few modifications as follows:

1. We modified the Makefile to use level 3 optimization when compiling the code with `gcc`. This ensures that the code is compiled to run faster.
2. We wrote a python version of `run_benchmarking.sh` available in `run_benchmark2.py`.
3. `run_benchmark2.py` is run with 3 required arguments (`start`, `end`, `step`) and 1 optional argument (`reps`).
4. We set the default number of repetitions to 20
5. In order to prevent bias in our experiments, we run the algorithms on randomly selected input sizes in the range of possible values.
6. We include the CSV generation process in `run_benchmarking.py`.

To ensure reproducibility, we provide our machine specifications below.

Machine Specification

CPU

AMD Ryzen 7 4800H with Radeon Graphics

Number of cores

8

Number of threads

16

RAM

24.7 GB

Data Analysis

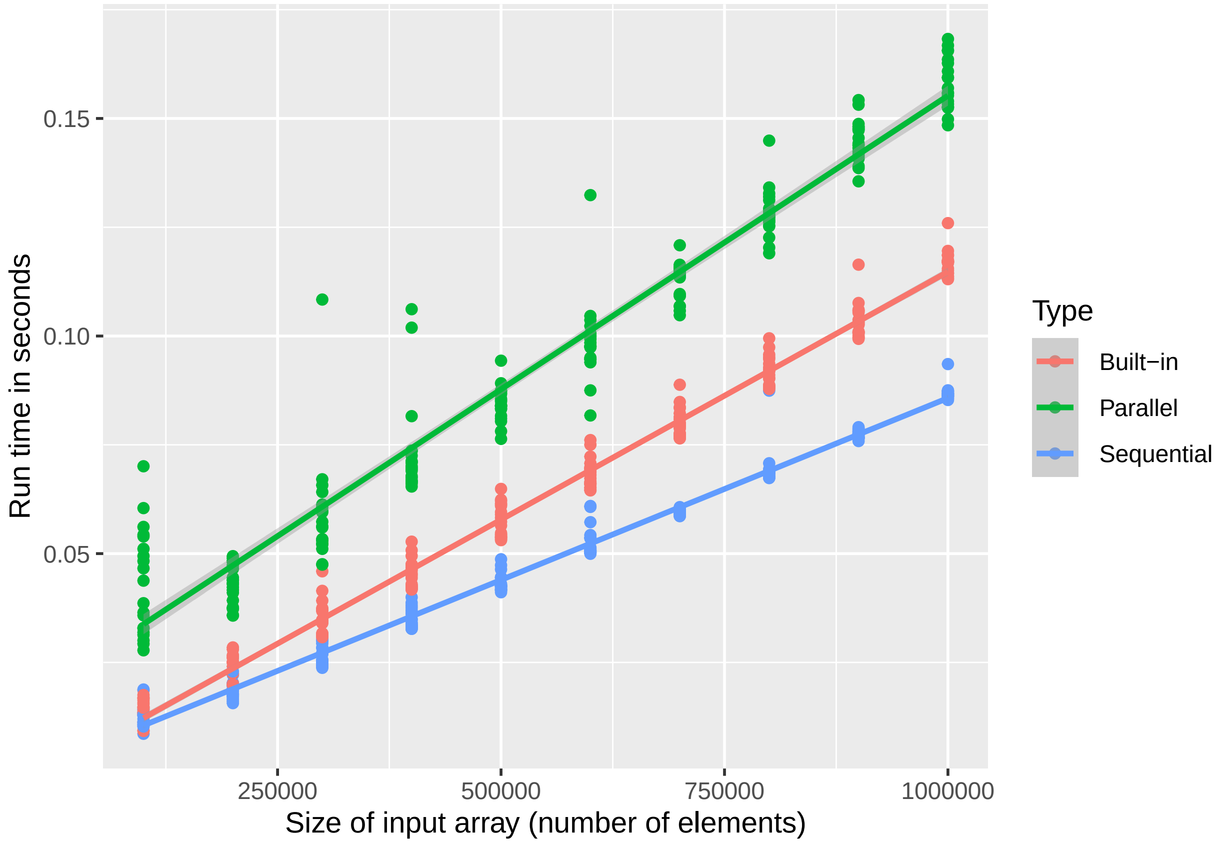
To get an idea of the relative performance of the sequential and parallel approaches, we run a first experiment with a `start` of size $1 \times e^5$, `end` of $1 \times e^6$ and a `step` size of $1 \times e^5$.

```
python scripts/run_benchmarking.py 1e5 1e6 1e5
```

The graph below shows the result of this experiment.

```
df1 <- read.csv("data/jimiolaniyan_2021-11-25/measurements_1M.csv", sep=",")
new_df <- df1 %>% group_by(Size, Type)
x = labs(x = "Size of input array (number of elements)")
y = labs(y = "Run time in seconds")
ggplot(new_df, aes(x=Size, y=Time, color=Type))+ geom_point() + geom_smooth(method='lm')+
↪  x + y
```

```
## `geom_smooth()` using formula 'y ~ x'
```

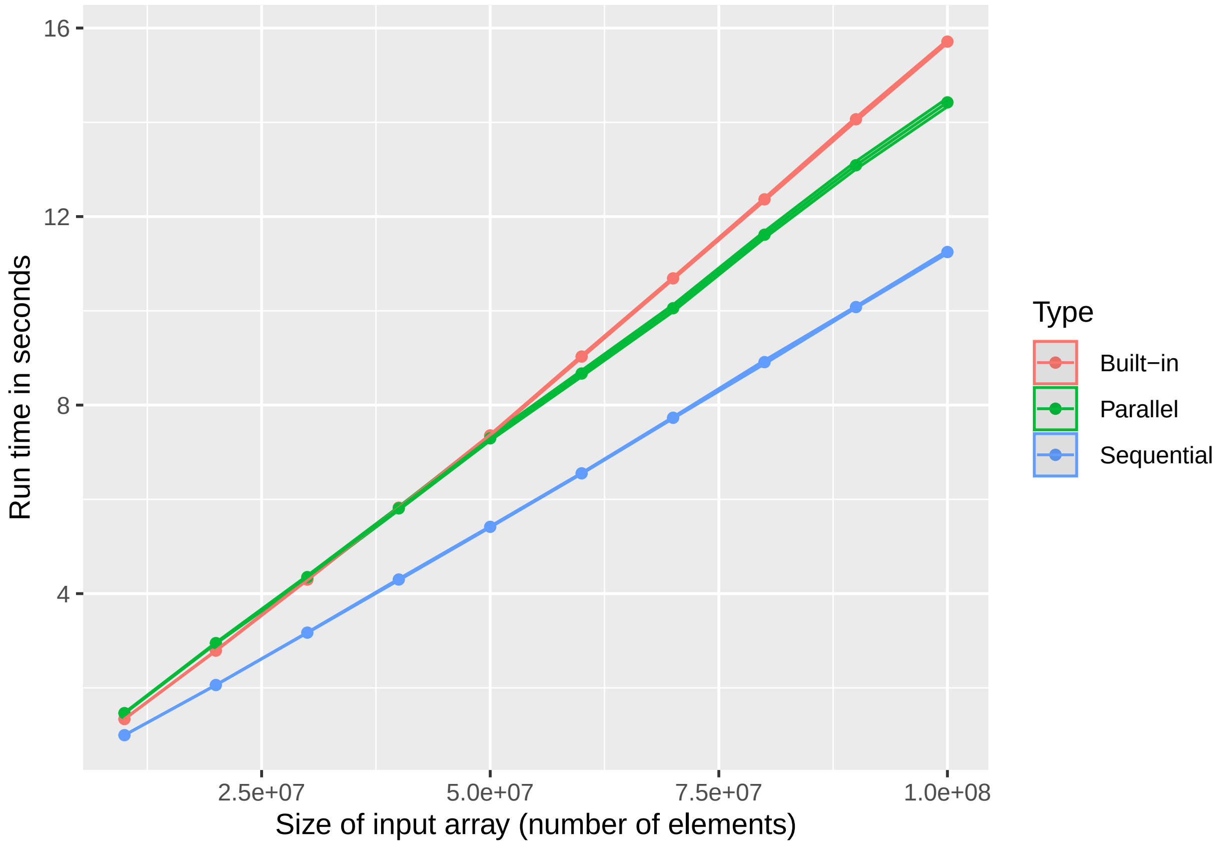


The graph shows that the Sequential algorithm is faster on average than the Parallel for input sizes up to 1 million. Next we try to get the point where Parallel quicksort is faster than the sequential version. For this we run the benchmark program with a `start` of size $1 \times e^7$, `end` of $1 \times e^8$ and a `step` size of $1 \times e^7$.

```
df2 <- read.csv("data/jimiolaniyan_2021-11-25/measurements_100M.csv", sep=",")
new_df <- df2 %>% group_by(Size, Type) %>% summarise(mean=mean(Time), sd=sd(Time), n=n())
```

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

```
x = labs(x = "Size of input array (number of elements)")
y = labs(y = "Run time in seconds")
ggplot(new_df, aes(x=Size, y=mean, color=Type))+ geom_point() +
↪  geom_ribbon(aes(ymin=mean-sd/sqrt(n), ymax=mean+sd/sqrt(n)), alpha=0.1) + geom_line()
↪  + x + y
```



```
#ggplot(new_df, aes(x=Size, y=mean, color=Type))+ geom_point() +
↪  geom_smooth(method='lm')+ x + y
```

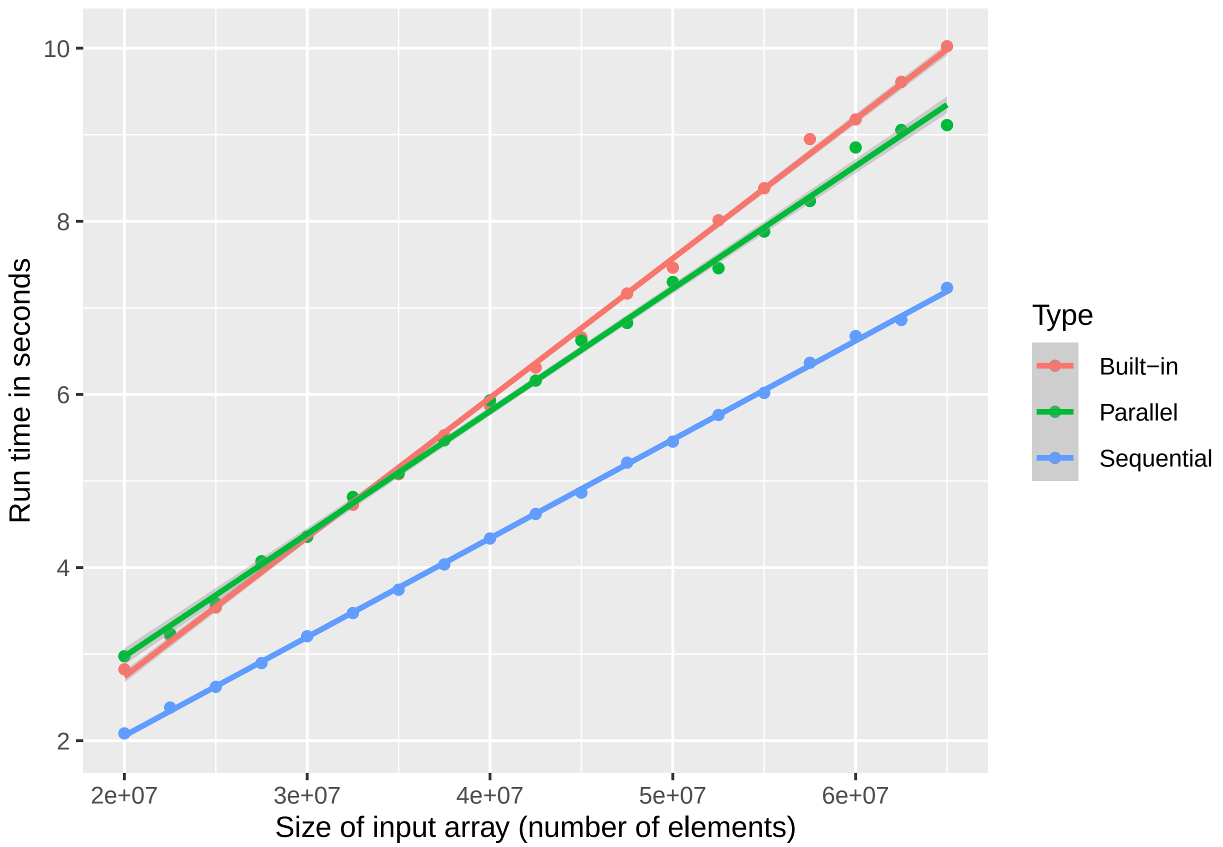
At this point, we observe that the parallel version runs faster from around a size of $5 \times e^7$ but still remains slower than the Sequential version. Below, we analyse the range $2 \times e^7$ and $7 \times e^7$ with a step size of $0.5 \times e^6$.

```
df3 <- read.csv("data/jimiolaniyan_2021-11-25/measurements_20M_70M.csv", sep=",")
new_df <- df3 %>% group_by(Size, Type) %>% summarise(mean=mean(Time), sd=sd(Time), n=n())
```

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

```
x = labs(x = "Size of input array (number of elements)")
y = labs(y = "Run time in seconds")
#ggplot(new_df, aes(x=Size, y=mean, color=Type))+ geom_point() +
↪  geom_ribbon(aes(ymin=mean-sd/sqrt(n), ymax=mean+sd/sqrt(n)), alpha=0.1) + geom_line()
↪  + x + y
ggplot(new_df, aes(x=Size, y=mean, color=Type))+ geom_point() + geom_smooth(method='lm')+
↪  x + y
```

```
## `geom_smooth()` using formula 'y ~ x'
```



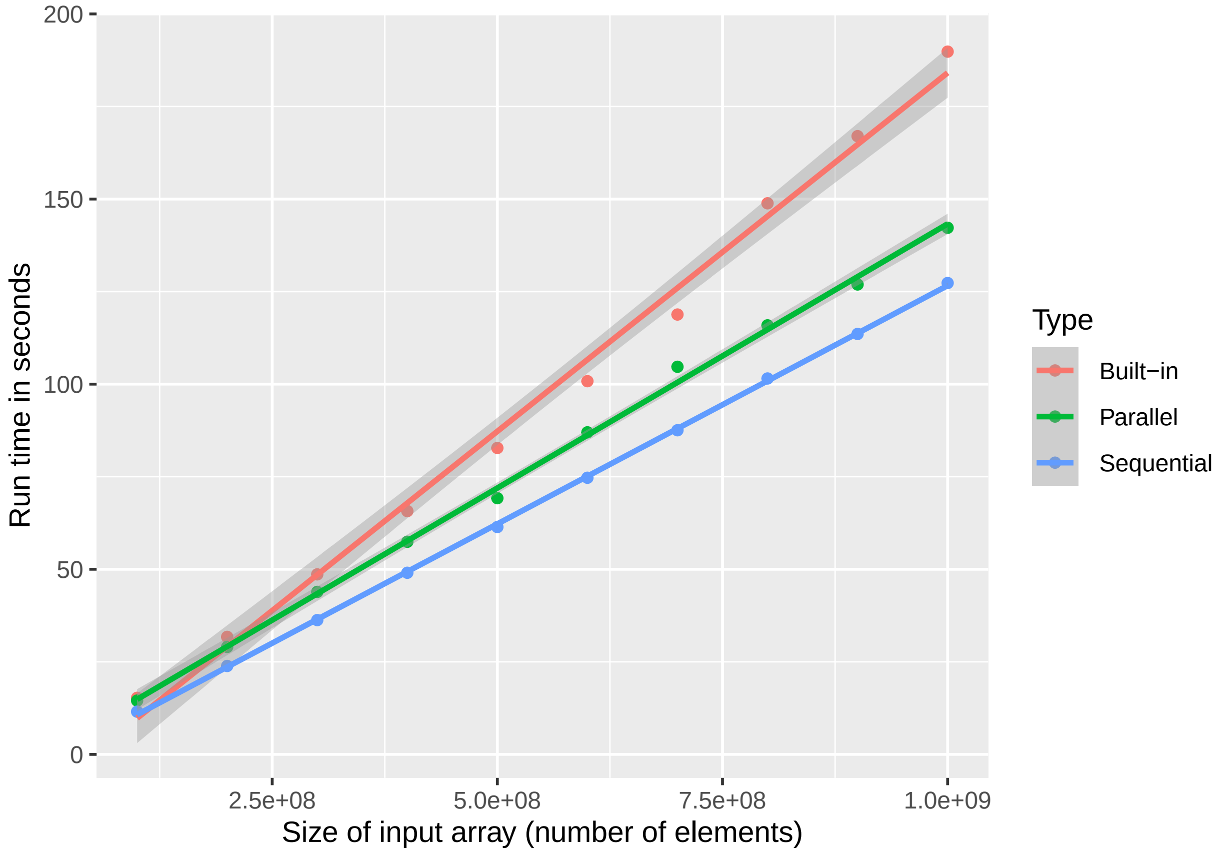
Here we see that the Parallel quicksort becomes better than the built-in sorting method after $3 \times e^7$ input size. Now we increase the range of values to between $1 \times e^8$ and $1 \times e^9$. The graphs if provided below.

```
df4 <- read.csv("data/jimiolaniyan_2021-11-24/measurements_1B.csv", sep=",")
new_df <- df4 %>% group_by(Size, Type) %>% summarise(mean=mean(Time), sd=sd(Time), n=n())
```

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

```
x = labs(x = "Size of input array (number of elements)")
y = labs(y = "Run time in seconds")
#ggplot(new_df, aes(x=Size, y=mean, color=Type))+ geom_point() +
↪  geom_ribbon(aes(ymin=mean-sd/sqrt(n), ymax=mean+sd/sqrt(n)), alpha=0.1) + geom_line()
↪  + x + y
ggplot(new_df, aes(x=Size, y=mean, color=Type))+ geom_point() + geom_smooth(method='lm')+
↪  x + y
```

```
## `geom_smooth()` using formula 'y ~ x'
```



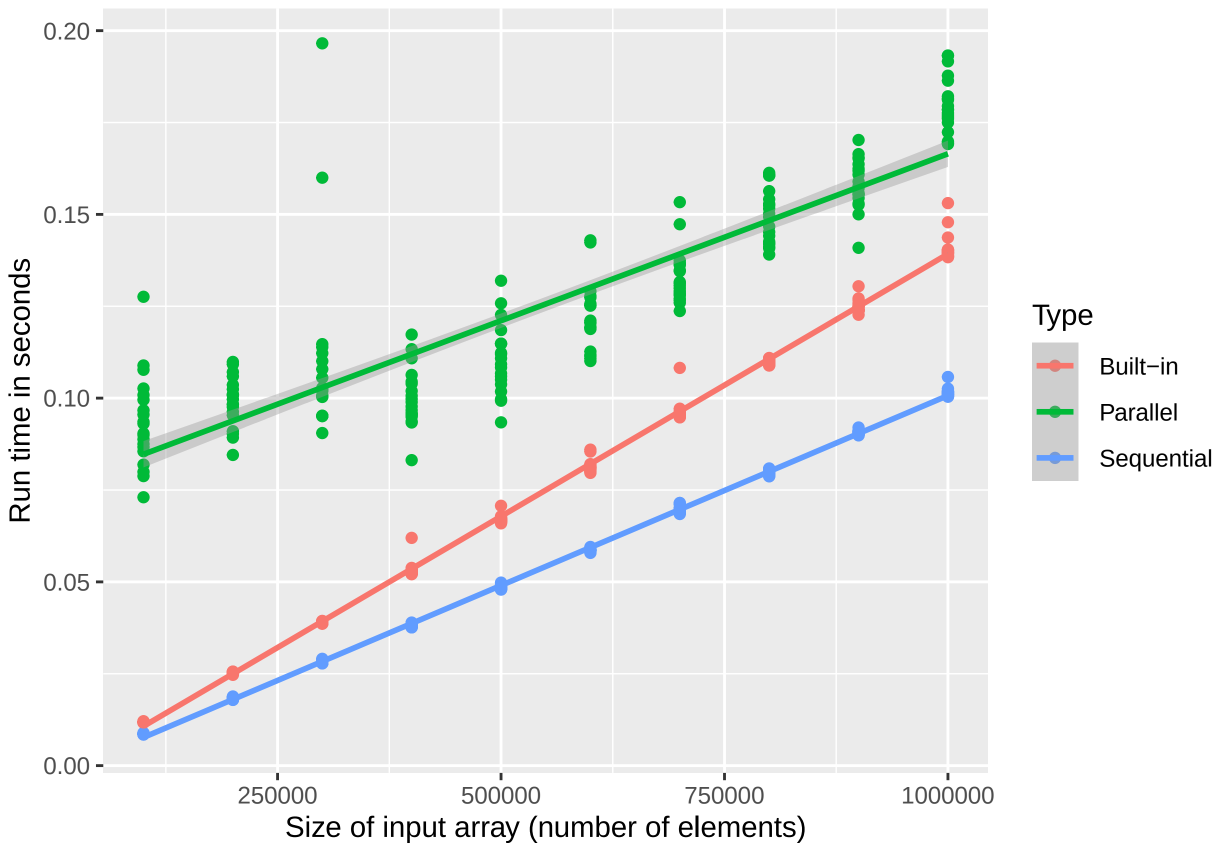
Although the parallel quicksort is not faster than its sequential counterpart, it get's closer to being faster. Nevertheless, at this point, we could not run experiments with larger input sizes as we constantly exceed the memory capacity of our machine. Following the trend of the graphs, we believe that the parallel sort should become faster than the sequential version for much higher values.

Execution on a less powerful machine

The same experiments can be performed on a different machine, in order to analyse if the detected trends are universal or machine-dependent. In particular, the CPU is an Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz. Windows 11 is the main OS on the machines, but parallel quicksort is compiled and executed on Ubuntu 20.04, running in the Windows Linux Subsystem with 4 GB of RAM. These parameters and the experiment conditions are tracked in the metadata of log files. All other programs are shut down and network is disabled while running the experiments.

```
df1 <- read.csv("data/LAPTOP-126V4913_2021-12-01/measurements_1M.csv", sep=",")
new_df <- df1 %>% group_by(Size, Type)
x = labs(x = "Size of input array (number of elements)")
y = labs(y = "Run time in seconds")
ggplot(new_df, aes(x=Size, y=Time, color=Type))+ geom_point() + geom_smooth(method='lm')+
↪  x + y
```

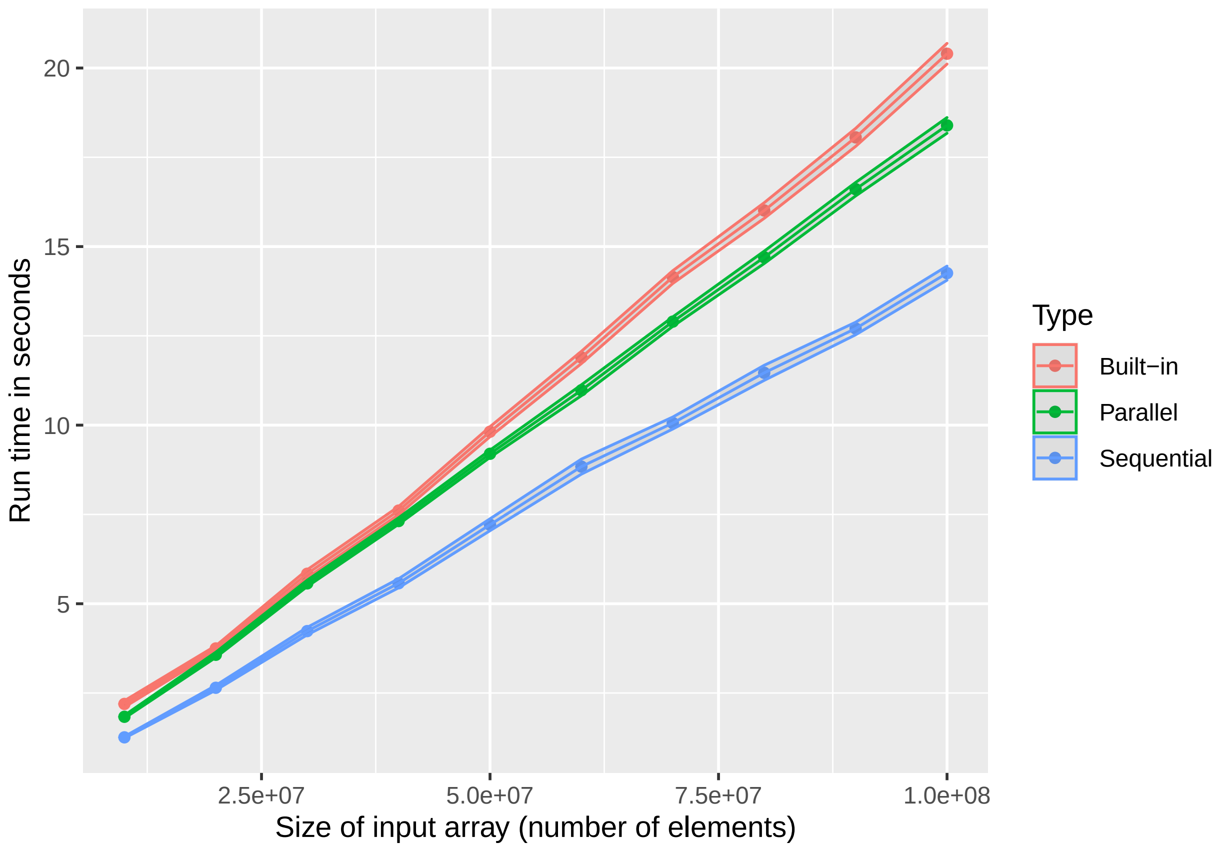
```
## `geom_smooth()` using formula 'y ~ x'
```



```
df2 <- read.csv("data/LAPTOP-126V4913_2021-12-02/measurements_100M.csv", sep=",")
new_df <- df2 %>% group_by(Size, Type) %>% summarise(mean=mean(Time), sd=sd(Time), n=n())
```

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

```
x = labs(x = "Size of input array (number of elements)")
y = labs(y = "Run time in seconds")
ggplot(new_df, aes(x=Size, y=mean, color=Type))+ geom_point() +
↪  geom_ribbon(aes(ymin=mean-sd/sqrt(n), ymax=mean+sd/sqrt(n)), alpha=0.1) + geom_line()
↪  + x + y
```



```
#ggplot(new_df, aes(x=Size, y=mean, color=Type))+ geom_point() +
↪  geom_smooth(method='lm')+ x + y
```

Notes

Metadatas are added to log files as Extended file attributes through Python's `os.setxattr()` function. It should be possible to retrieve them in R through the `xattrs` package. Unfortunately, I am using on a Windows machine and was not able to make it work. However, they are correctly read with Python's `os.setxattr()`.

We would then like to highlight a few directions we could take in the future:

1. We need to investigate the code to make sure that it actually creates enough threads to take advantage of full parallelism.