LAB REPORT

PARALLEL PROGRAMMING : LAB 01

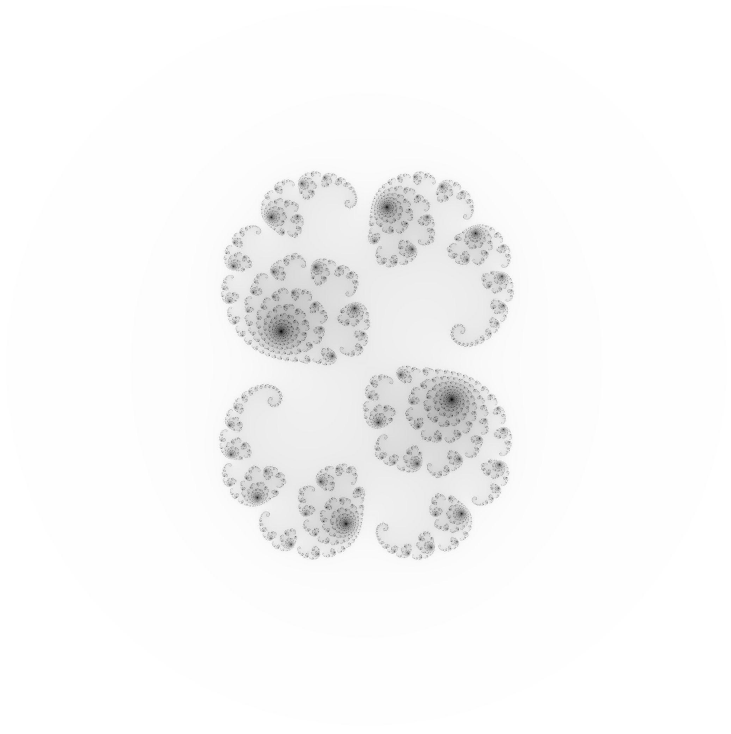
Autors :

Etienne CORREGE

Arnaud SAINT CIRGUE

Teacher :

M.HAMELIN (ING5 SE Gr02)



We certify that this work is original, the result of collaborative effort by the duo, and has been independently written.

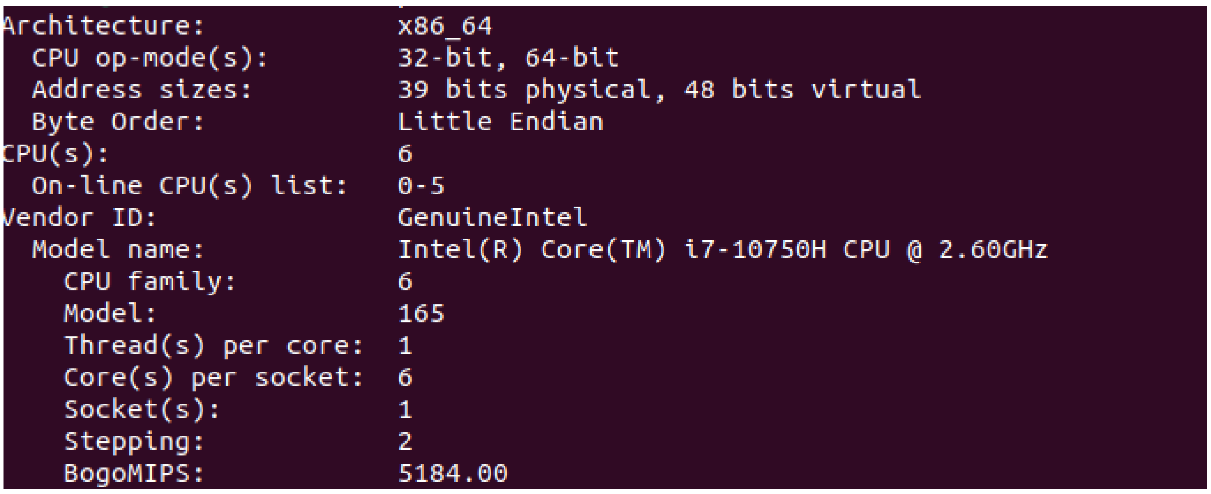
Paris, 23/10/2023

Table of contents

0. Know your hardware

Q1 : What is your OS/hypervisor system configuration?
How many CPU physical and logical cores does your PC run? How much RAM memory (in GiB)?
Use the commands cat /proc/cpuinfo, and cat /proc/meminfo to answer.

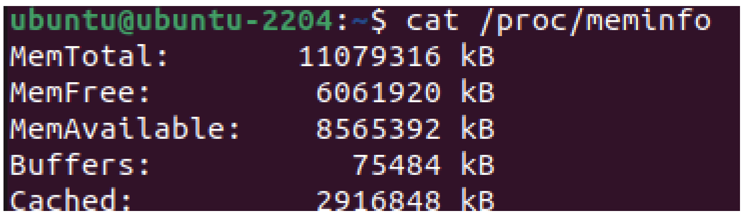
- Configuration VM (Arnaud Saint-Cirgue)



```
rchitecture:            x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          39 bits physical, 48 bits virtual
  Byte Order:             Little Endian
PU(s):                    6
  On-line CPU(s) list:    0-5
endor ID:                 GenuineIntel
  Model name:             Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
    CPU family:           6
    Model:                165
    Thread(s) per core:   1
    Core(s) per socket:   6
    Socket(s):            1
    Stepping:             2
    BogoMIPS:             5184.00
```

Figure 0.1 : cpu configuration on Arnaud Saint-Cirgue laptop

The computer has 2 CPU sockets. Each CPU socket has 6 physical cores. Hence, the computer has 6 physical cores in total. Each physical CPU core can run 1 thread. These threads are the core's logical capabilities. The total number of logical cores = CPU sockets × physical cores per socket × threads per physical core. Therefore, the computer has 1 × 6 × 1 = 6 logical cores in total.

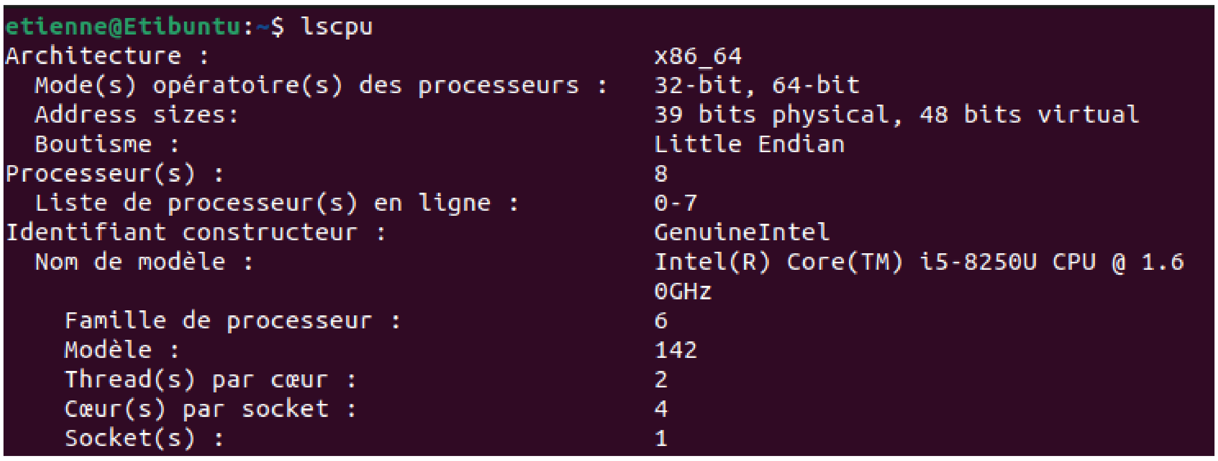


```
ubuntu@ubuntu-2204:~$ cat /proc/meminfo
MemTotal:       11079316 kB
MemFree:         6061920 kB
MemAvailable:    8565392 kB
Buffers:           75484 kB
Cached:          2916848 kB
```

Figure 0.2 : memory configuration on Arnaud Saint-Cirgue's laptop

By using the "cat /proc/meminfo" (Figure 0.2), we have a total of 11GB RAM.

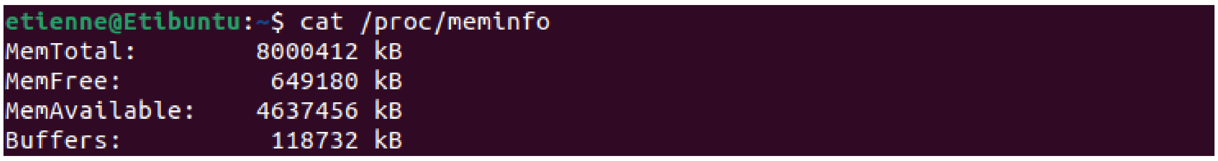
- Configuration Dual Boot (Etienne Corrège)



```
etienne@Etibuntu:~$ lscpu
Architecture :                                x86_64
  Mode(s) opératoire(s) des processeurs :     32-bit, 64-bit
  Address sizes:                              39 bits physical, 48 bits virtual
  Boutisme :                                  Little Endian
Processeur(s) :                               8
  Liste de processeur(s) en ligne :           0-7
Identifiant constructeur :                    GenuineIntel
  Nom de modèle :                             Intel(R) Core(TM) i5-8250U CPU @ 1.6
                                              0GHz
    Famille de processeur :                   6
    Modèle :                                  142
    Thread(s) par cœur :                      2
    Cœur(s) par socket :                      4
    Socket(s) :                               1
```

Figure 0.3 : cpu configuration on Etienne Corrège laptop

With the same method, we have a total of 8 logical cores (Figure 0.3).



```
etienne@Etibuntu:~$ cat /proc/meminfo
MemTotal:        8000412 kB
MemFree:          649180 kB
MemAvailable:    4637456 kB
Buffers:          118732 kB
```

Figure 0.4 : memory configuration on Etienne Corrège's laptop

By using the "cat /proc/meminfo" (Figure 0.4), we have a total of 8GB RAM.

1. Memory organization in C

Q2 : From the code mem.c, identify what segment the following variables/data will be stored in.

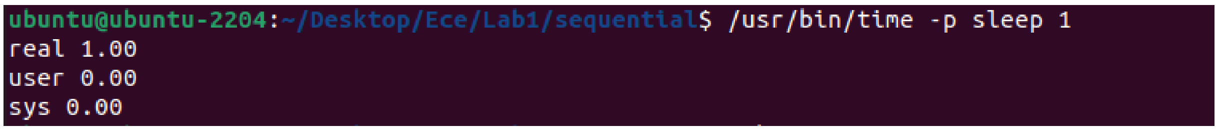
	Static	Stack	Heap
f		X	
g	X		
n		X	
res		X	
cnt	X		
str			X
str[0…]			X
str2		X	
str2[0…]		X	

Figure 1.1 : storage of the variables/data from mem.c

2. Performance measurements

1. Sequential performance

Q3 : Look up the manual to understand the meaning of the real, user and sys fields.

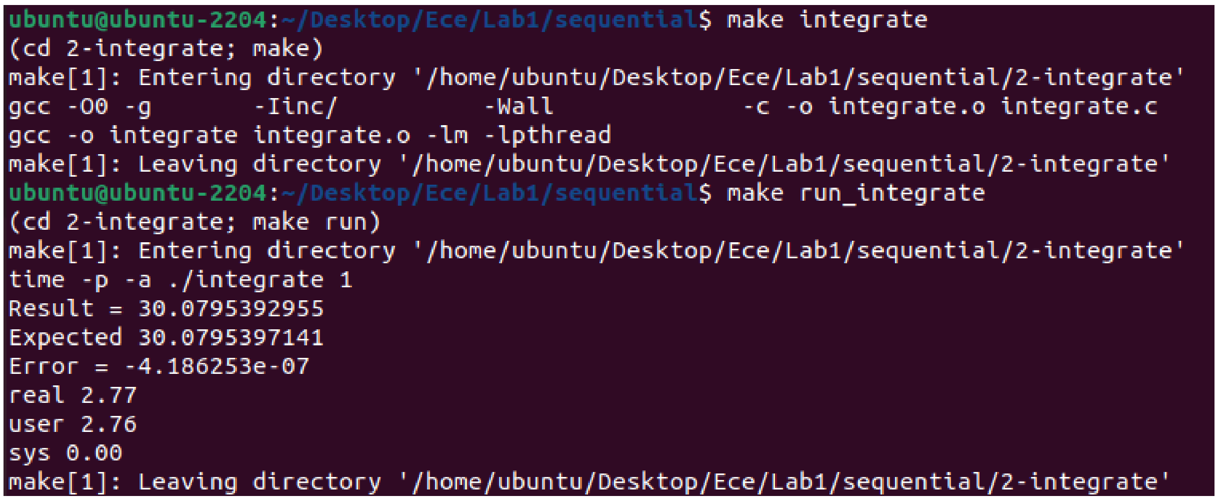


```
ubuntu@ubuntu-2204:~/Desktop/Ece/Lab1/sequential$ /usr/bin/time -p sleep 1
real 1.00
user 0.00
sys 0.00
```

<u>*Figure 2.1.1 : timings with sleep function*</u>

The "real" variable represents the total execution time of a command, in this case 1 second because the processor is made to "sleep" for 1 second. It is therefore normal for the user variable, which represents the execution time of the processor, to be equal to 0. This is also the case for the sys variable, which represents system operations (input/output, etc.), which here is equal to 0.

Q4: What is the value of: real, user, sys, $\frac{user+sys}{real}$?



```
ubuntu@ubuntu-2204:~/Desktop/Ece/Lab1/sequential$ make integrate
(cd 2-integrate; make)
make[1]: Entering directory '/home/ubuntu/Desktop/Ece/Lab1/sequential/2-integrate'
gcc -O0 -g       -Iinc/           -Wall             -c -o integrate.o integrate.c
gcc -o integrate integrate.o -lm -lpthread
make[1]: Leaving directory '/home/ubuntu/Desktop/Ece/Lab1/sequential/2-integrate'
ubuntu@ubuntu-2204:~/Desktop/Ece/Lab1/sequential$ make run_integrate
(cd 2-integrate; make run)
make[1]: Entering directory '/home/ubuntu/Desktop/Ece/Lab1/sequential/2-integrate'
time -p -a ./integrate 1
Result = 30.0795392955
Expected 30.0795397141
Error = -4.186253e-07
real 2.77
user 2.76
sys 0.00
make[1]: Leaving directory '/home/ubuntu/Desktop/Ece/Lab1/sequential/2-integrate'
```

<u>*Figure 2.1.2 : timings on integrate code*</u>

After executing the command(Figure 2.1.2), we obtain the following values :

Real = 2,77

User = 2,76

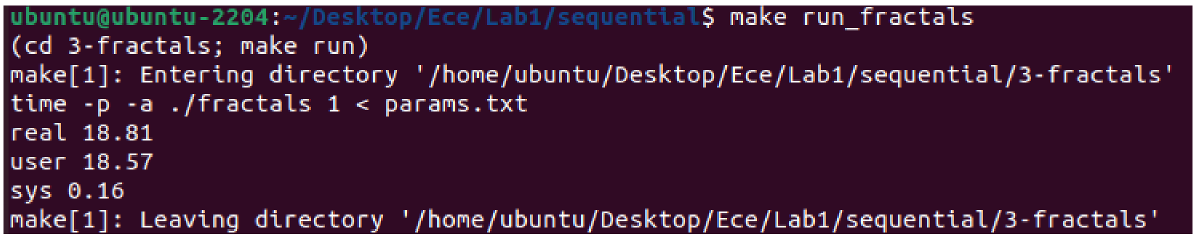
Sys = 0,00

Finally, $\frac{user+sys}{real} = \frac{2,76+0}{2,77} = 0,996$ (rounded to the thousandth)

Q5: What does the ratio $\frac{user+sys}{real}$ *represent ?*

The ratio is a measure of CPU usage during execution of the command compared to the actual elapsed time. It represents the efficiency of CPU use in relation to the program. The higher the ratio, the more efficiently the command used the CPU during execution.

Q6 : What is the value of: real, user, sys, $\frac{user+sys}{real}$ *?*



```
ubuntu@ubuntu-2204:~/Desktop/Ece/Lab1/sequential$ make run_fractals
(cd 3-fractals; make run)
make[1]: Entering directory '/home/ubuntu/Desktop/Ece/Lab1/sequential/3-fractals'
time -p -a ./fractals 1 < params.txt
real 18.81
user 18.57
sys 0.16
make[1]: Leaving directory '/home/ubuntu/Desktop/Ece/Lab1/sequential/3-fractals'
```

Figure 2.1.3 : execution time on fractals code

After executing the command(Figure 2.1.3), we obtain the following values :

Real = 18.81

User = 18.57

Sys = 0.16

This time the ratio $\frac{user+sys}{real} = \frac{18{,}57+0{,}16}{18{,}81} = 0,995$

Q7 : Why is the ratio $\frac{user+sys}{real}$ *different from Q5?*

Here, the ratio is different because we are generating files (the fractals named "julia XXX") which take up system resources and therefore add to execution time.

2. Parallel programming in practice!

Q8 : Parallelize integrate.c using Pthreads. Read a number of threads from command line arguments (argc, argv). Verify that the outcome is the same (to at least 0.001 precision)!

After modifying the integrated code with the threads implementation, we obtain the results below by increasing the number of threads from 1 to 20.

nb_thread	real	user	sys	nb_thread	real	user	sys
1	0:02.85	2.82	0.02	11	0:00.60	3.26	0
2	0:01.42	2.8	0.01	12	0:00.66	3.2	0.04
3	0:01.03	2.94	0.02	13	0:00.70	3.19	0.01
4	0:00.96	3.03	0.02	14	0:00.71	3.24	0
5	0:00.88	3.43	0.02	15	0:00.70	3.49	0.04
6	0:00.68	3.23	0.01	16	0:00.68	3.41	0.01
7	0:00.68	3.05	0.01	17	0:00.74	3.33	0.04
8	0:00.59	3.09	0	18	0:00.70	3.41	0.02
9	0:00.67	3.12	0	19	0:00.63	3.26	0.01
10	0:00.68	3.08	0.01	20	0:00.65	3.25	0

Figure 2.2.1 : table of integrate.c execution time sorted by the number of threads

We can see on *Figure 2.2.1* that from 8 threads upwards, the "real" and "user" times stabilize around a certain value. We can therefore deduce that the number of threads has a limit when it comes to optimizing code execution time.

Q9 : Draw a graph of latency (real), as a function of thread number

Based on the previous question's result, we can draw a graph of the latency as a function of the number of threads.

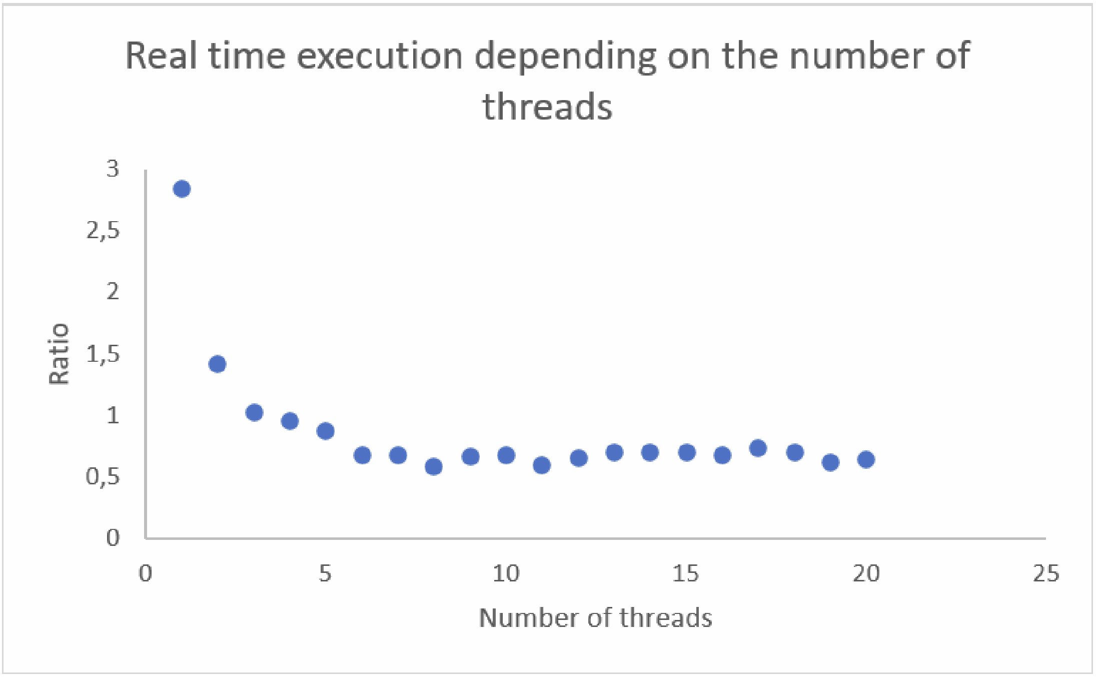


Figure 2.2.2 : graph of integrate.c execution time versus number of threads

It looks like the function follows a power law, and tends to a value close to 0 but not 0 because the value does not change so much after 7 or 8 threads.

Q10 : Draw a graph of the ratio $\frac{real_{seq}}{real(n)}$ as a function of thread number ($real_{seq}$ being the value measured in Q4 above). What is the maximum value? Explain what happens when $n \geq$ your PC's core count.

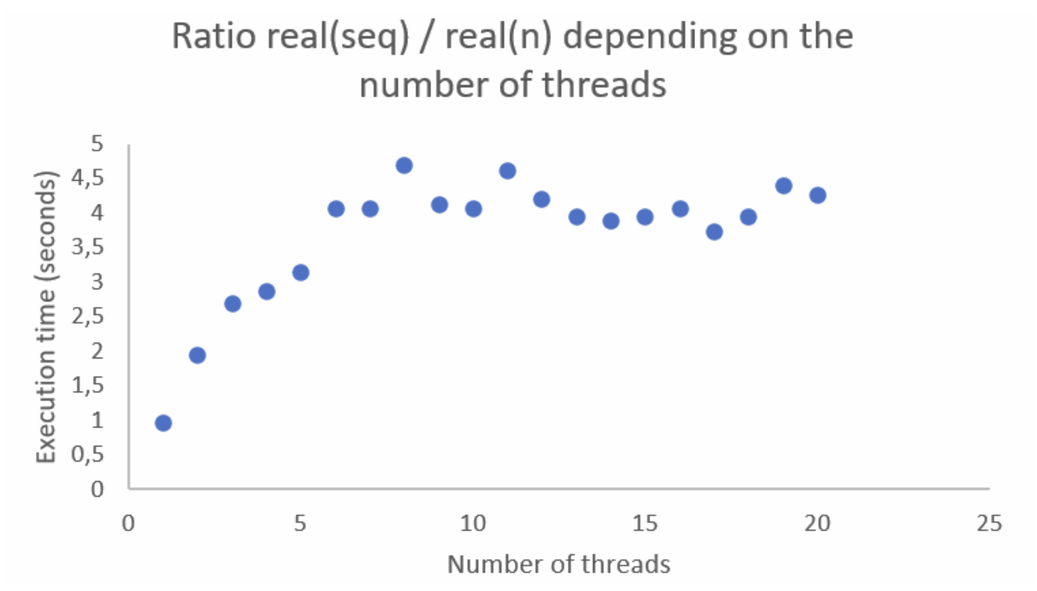


Figure 2.2.3 : graph representing the real(seq)/real(n) ratio of integrate.c

The maximum value on the graph corresponds when we use 8 threads and gives a ratio of 4,69. When the number of threads used exceeds the number of processors available (in this case 6 processors), the ratio tends to stagnate at around 3,5, so the machine can execute on average 4 times faster than sequential execution when we use the parallel programmation correctly. This is a case of parallel execution overhead. It seems that it is interesting to use threads until the number of threads exceeds the number of processors around 8 in this example.

Q11 : Parallelize fractal.c using Pthreads similarly. Verify that images are correct!

After modifying the integrated code with the threads implementation, we obtain the results below by increasing the number of threads from 1 to 20.

nb_thread	**real**	**user**	**sys**	**nb_thread**	**real**	**user**	**sys**
1	0:20.13	19.13	0.3	11	0:07.67	20.54	0.47
2	0:10.86	19.76	0.27	12	0:06.80	20.68	0.47
3	0:16.52	19.87	0.35	13	0:06.84	20.33	0.4
4	0:10.56	20.23	0.34	14	0:06.39	20.58	0.37
5	0:12.21	19.77	0.38	15	0:06.86	21.41	0.53
6	0:08.82	19.69	0.42	16	0:06.85	22.38	0.5
7	0:10.13	20.49	0.36	17	0:06.30	21.58	0.47
8	0:08.42	21.09	0.38	18	0:06.10	21.69	0.46
9	0:08.76	20.73	0.44	19	0:05.92	20.87	0.51
10	0:07.55	20.8	0.44	20	0:05.61	20.28	0.5

Figure 2.2.4 : table of fractals.c execution time sorted by the number of threads

Q12 : Draw the graph of latency (real) as a function of thread number

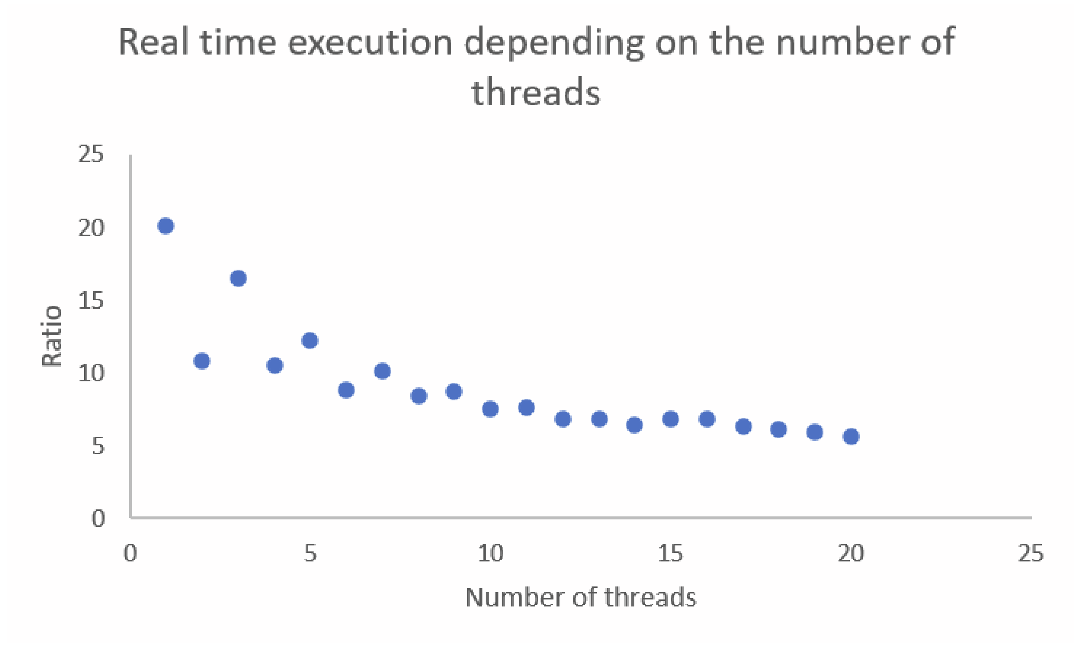


Figure 2.2.5 : graph of fractals.c execution time versus number of threads

We notice in the *figure 2.2.5* that the more threads we use, the more we tend towards a value closer to 5. The reduction in real time is significant (reduction in seconds) up to the 6-7th thread.

Q13 : Draw a graph of the ratio $\frac{real_{seq}}{real(n)}$ *as a function of thread number (*$real_{seq}$ *for fractal was measured in Q6).*

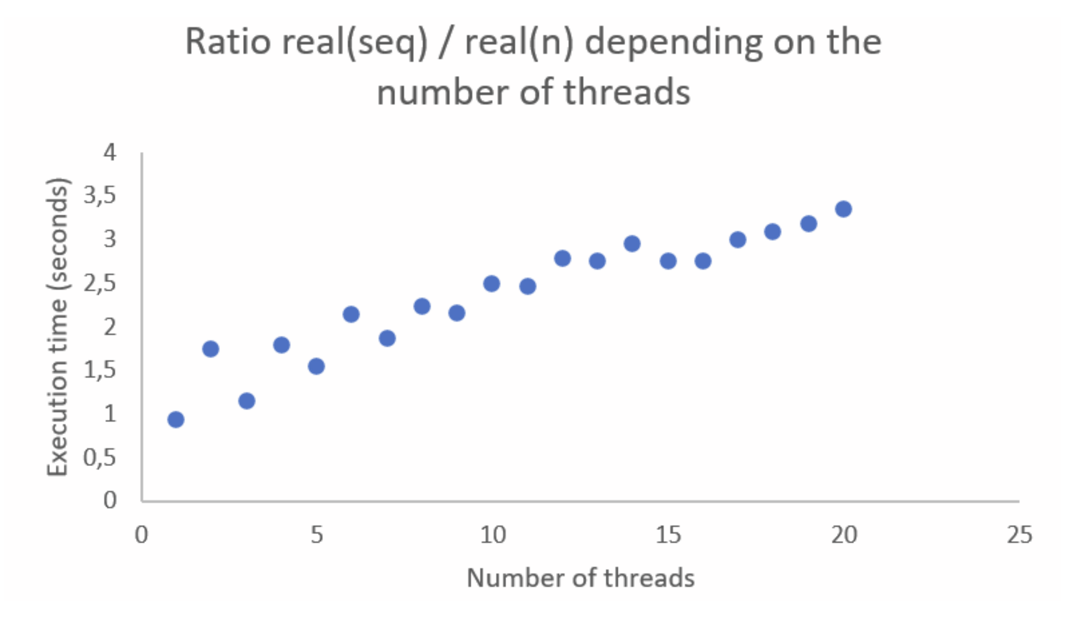


Figure 2.2.6 : graph representing the real(seq)/real(n) ratio of integrate.c

Q14 : Based on your experiments with integrate and fractal, what does $\frac{real_{seq}}{real(n)}$ *represent? Explain.*

Based on what we have observed in the two examples, the ratio “real(seq) / real(n)” measures the extent to which parallel execution is faster or slower than sequential execution for a given task.

When the result is lower than 1, it means that parallel execution is slower than sequential execution. When the result is greater than 1, This means that parallel execution is faster than sequential execution, and this is what we are looking for.

So this ratio is an effective way of determining whether to use sequential programming or parallel execution and also this proves that when the number of threads used exceeds the number of processors, parallel programming reaches its maximum optimisation.