TEAM SUMMIT:
Week 03:

Number 01: (monte-carlo approx.)

- Implement parallel versions of the Monte Carlo Pi approximation of assignment 02 using OpenMP. In total, three different versions using the following OpenMP constructs should be provided:
  - `critical` section
  - `atomic` statement
  - `reduction` clause

The C Code is provided. But here a quick overview what is special about every single one.

Atomic:
atomic only supports a restricted set of expressions, all those expressions happen inseparable, either it happens completely
atomic operation has way less overhead than critical,

Critical:
a critical block means that in this specific block, only one thread can be at a time
means that, if thread 1 is in the critical block, all the other threads need to wait until it has finished
this synchronization means a lot of overhead, so like you can see later it is way the slowest one

Reduction:
reduction gets translated to a private variable which is incremented in the loop
atomic operation of reduction gets done at the end of the loop in an atomic clause

- Benchmark your parallel implementations with 1, 2, 4, and 8 threads on LCC2 using n=500,000,000 using OpenMP's time measurement function. What can you observe? How do those results compare to your measurements in Assignment 02?

1 Thread:

```
[[cb761113@login.lcc2 cb761113]$ OMP_NUM_THREADS=1 ./monte_reduction
Pi: 3.141587
 time: 15.5550 seconds
[[cb761113@login.lcc2 cb761113]$ OMP_NUM_THREADS=1 ./monte_atomic
Pi: 3.141587
 time: 23.7378 seconds
[[cb761113@login.lcc2 cb761113]$ OMP_NUM_THREADS=1 ./monte_critical
Pi: 3.141587
 time: 31.6701 seconds
```

2 Threads:

```
[[cb761113@login.lcc2 cb761113]$ OMP_NUM_THREADS=2 ./monte_critical
Pi: 3.141593
 time: 115.2548 seconds
[[cb761113@login.lcc2 cb761113]$ OMP_NUM_THREADS=2 ./monte_atomic
Pi: 3.141593
 time: 41.8653 seconds
[[cb761113@login.lcc2 cb761113]$ OMP_NUM_THREADS=2 ./monte_reduction
Pi: 3.141593
 time: 7.7838 seconds
```

4 Threads:

```
[[cb761113@login.lcc2 cb761113]$ OMP_NUM_THREADS=4 ./monte_reduction
Pi: 3.141587
 time: 3.8925 seconds
[[cb761113@login.lcc2 cb761113]$ OMP_NUM_THREADS=4 ./monte_atomic
Pi: 3.141587
 time: 40.1372 seconds
[[cb761113@login.lcc2 cb761113]$ OMP_NUM_THREADS=4 ./monte_critical
Pi: 3.141587
 time: 222.1616 seconds
```

8 threads:

```
[[cb761113@login.lcc2 cb761113]$ OMP_NUM_THREADS=8 ./monte_reduction
Pi: 3.141544
 time: 2.0312 seconds
[[cb761113@login.lcc2 cb761113]$ OMP_NUM_THREADS=8 ./monte_atomic
Pi: 3.141544
 time: 54.5573 seconds
[[cb761113@login.lcc2 cb761113]$ OMP_NUM_THREADS=8 ./monte_critical
Pi: 3.141544
 time: 410.8716 seconds
```

How do they differ from last time? (Assignment 02)

n = 500.000.000 iterations		
	pi	time [sec.]
NUM_THREADS=1	0,134218	55,77
NUM_THREADS=2	0,134218	55,78
NUM_THREADS=3	0,402653	149,97
NUM_THREADS=4	0,536871	244,06
NUM_THREADS=5	0,671089	1514,73
NUM_THREADS=6	0,805306	1617,39
NUM_THREADS=7	0,939524	1986,72
NUM_THREADS=8	1,073742	1099,81

The yellow marked ones were the results from week02 from the pi approximation.
So, we had the same itertations#. So, this is can be ignored.

1 Thread:
OLD (55.77) vs
NEW 15.5(r) 23,7(a) 31,6(c)

2 Threads:
OLD (55.78) vs
NEW 7.7(r) 41,8(a) 115.2(c)

4 Threads:
OLD (244.06) vs
NEW 3.8 40.1. 222.16

8 Threads:
OLD (1099.81) vs
NEW 2.2(r) 54.5 (a) 410.8(c)

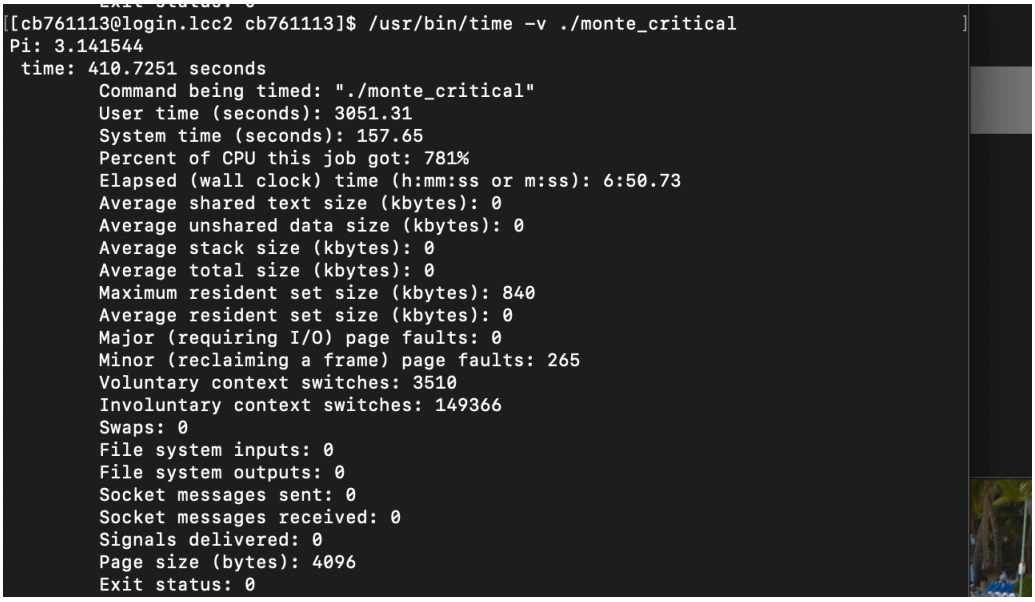
ALL different parallel versions are way faster than the sequential one from the previous sheet. But the critical one is by far the slowest one. At two threads it is even slower than the sequential program from the previous exercise.

Possible Reason?
Because the critical section is a **section** so that there is way more payload in there.

Usr/bin/time

```
[[cb761113@login.lcc2 cb761113]$ /usr/bin/time -v ./monte_atomic
Pi: 3.141544
 time: 54.3238 seconds
        Command being timed: "./monte_atomic"
        User time (seconds): 412.37
        System time (seconds): 0.03
        Percent of CPU this job got: 759%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:54.32
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 840
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 265
        Voluntary context switches: 16
        Involuntary context switches: 14988
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

```
[[cb761113@login.lcc2 cb761113]$ /usr/bin/time -v ./monte_reduction
Pi: 3.141544
 time: 2.0451 seconds
        Command being timed: "./monte_reduction"
        User time (seconds): 15.59
        System time (seconds): 0.00
        Percent of CPU this job got: 761%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.04
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 836
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 264
        Voluntary context switches: 13
        Involuntary context switches: 629
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```



How do they differ?

User Time differences:
3051 – Critical (50.85)
15.59 Reduction
412.37 Atomic

Elapsed Time differences:
6:50.73 – critical
0:2.04 - reduced
00:54.32 Atomic

Resume: The user time is way smaller than the elapsed time. That is because the elapsed time that the CPU gets charged for the expression. User Time is the wall clock time.
What that means?
So, if elapsed time > user time would mean that the cpu would have to wait for some other operations. For example, we have 5.85(User Time Critical) vs 6.50 (Elapsed Time critical)

Does it match with the time measurement from the openmp measurement function?

From the usr/bin/time:
6:50.73 – critical
0:2.04 - reduced
00:54.32 Atomic

Vs.:

From openMp Function:
8 Threads:
OLD (1099.81) vs
NEW 2.2(reduced) 54.5 (atomic) 410.8(critical) (== 6.8s)

Resume:
It is exactly the same time which was needed before **for 8 threads**. So, they differ in general, but match exactly with the measurement before with 8 threads. But they differ in for example one thread included.