a.)

```c
C exc1_openmp4.c ×

Users > julianhotter > Desktop > C exc1_openmp4.c > ...
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

//large number that at least it needs 2 seconds
#define N 114748367


int main(){

    int valuetoIncrement = 0;


    double startTime = omp_get_wtime();

#pragma omp parallel for
for (int i = 0; i < N; i++){


#pragma omp atomic
valuetoIncrement++;

    }

double endTime = omp_get_wtime();
printf("time: %2.2f seconds\n", endTime-startTime);




        return EXIT_SUCCESS;
}

```

AM LCC2 (the job script uses 8.2.0) like in the task

```
[[cb761113@login.lcc2 ~]$ gcc -std=c99  exc1.c -o exc1  -fopenmp -O3
[[cb761113@login.lcc2 ~]$ ./exc1
 time: 10.92 seconds
```

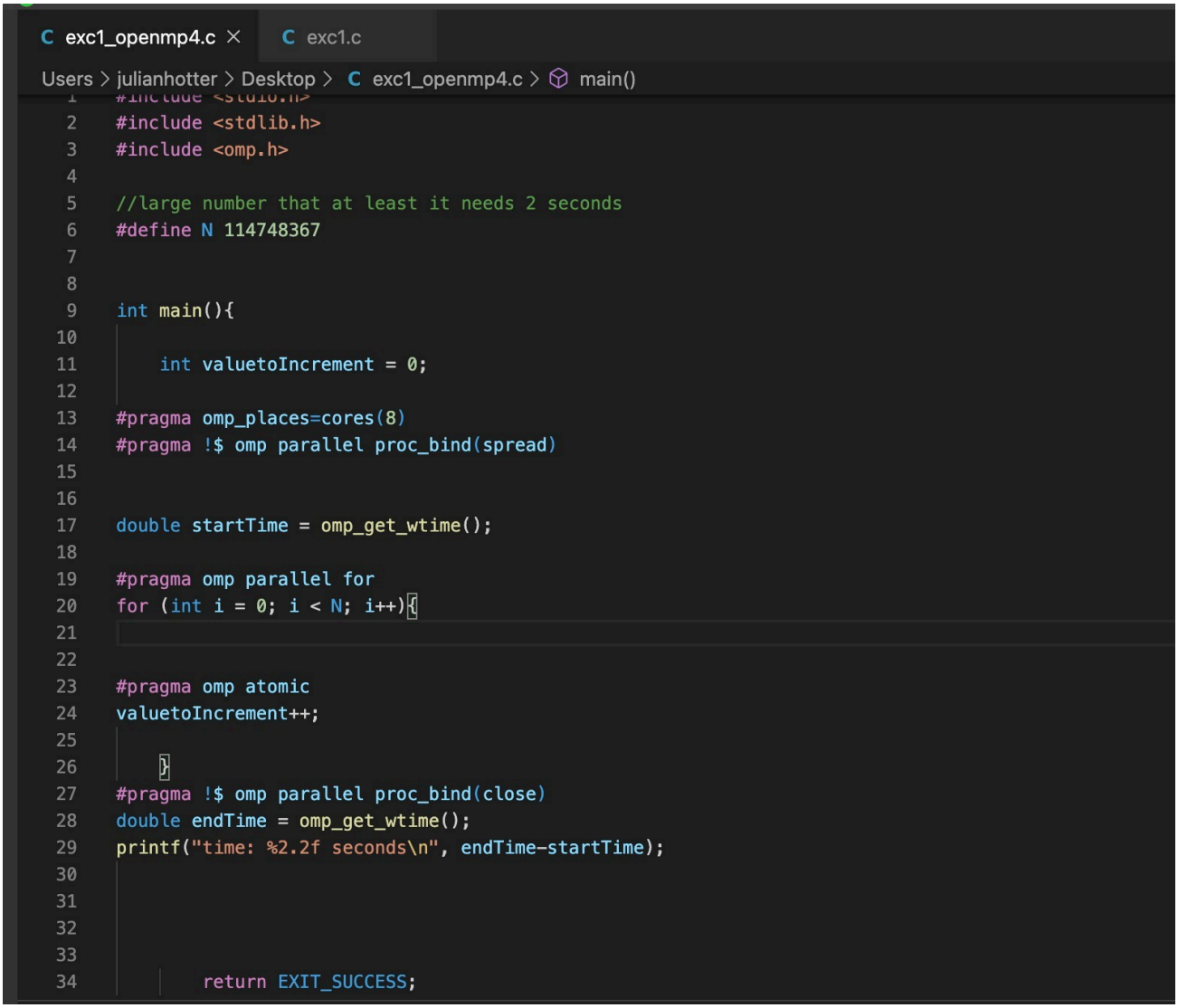
In contrast to this, i also run this on my local MacBook, but with gcc-10 because 8.2.0 had a problem with OpenMp. But got way better results.

Why?

Maybe because of the “newer” “better” compiler.

Now with OpenMP:

I Changed the code.



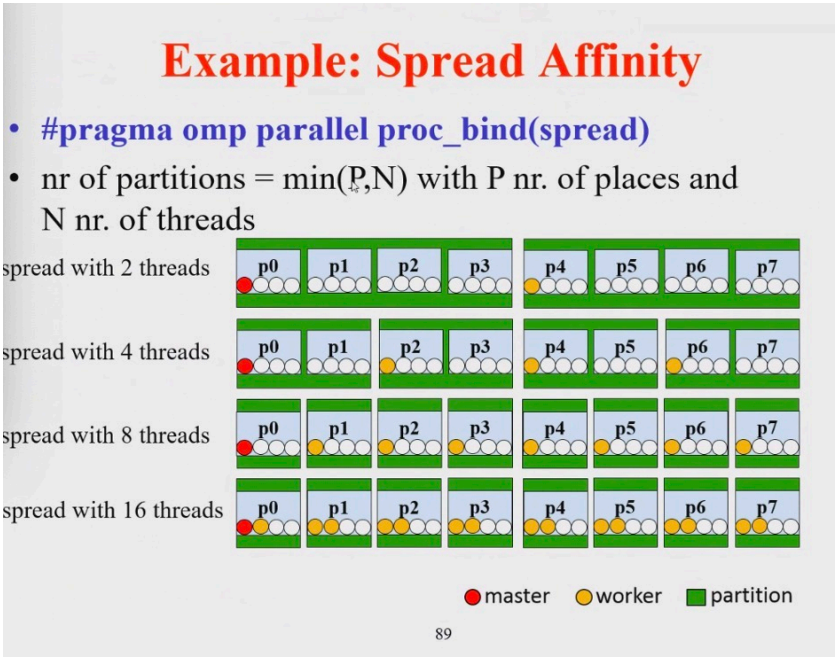
But theres only a slight difference between them both, to be sure I run them multiple times.
The exc1-better is the one with the Thread Affinity.

```
zid-vpns-15-69:desktop julianhotter$ OMP_NUM_THREADS=4 ./exc1-better
time: 3.33 seconds
zid-vpns-15-69:desktop julianhotter$ OMP_NUM_THREADS=4 ./exc1-better
time: 3.39 seconds
zid-vpns-15-69:desktop julianhotter$ OMP_NUM_THREADS=4 ./exc1-better
time: 3.38 seconds
zid-vpns-15-69:desktop julianhotter$ OMP_NUM_THREADS=4 ./exc1
time: 3.45 seconds
zid-vpns-15-69:desktop julianhotter$ OMP_NUM_THREADS=4 ./exc1
time: 3.39 seconds
zid-vpns-15-69:desktop julianhotter$ OMP_NUM_THREADS=4 ./exc1
time: 3.44 seconds
```

Proc bind as you can see is the clause.
The policy is spread, as mentioned in the lecture

That means openmp internally uses maximal spreaded (different) places.

Like in the lecture:



I Also tried the other policies like master and close but there was the same, the results differ only a little bit, which could be also some other stuff going on.

Solution – From Benchmarking my CPU Setting with Spread Affinity.
Maybe a possible reason would be that my laptop automatically shares among the cores and uses multithreading. Or is the change as small because it does not differ significantly because the CPU is anyway that fast.