* Makefile command: make

code inside makefile:

CC = mpicc

CFLAGS = -std=c99

ID = -I/software/FFTW/3.3.3-RH7/include

LD = -L/software/FFTW/3.3.3-RH7/lib

all: cgl

cgl: cgl.o

        $(CC) $(CFLAGS) $(ID) -o cgl $(LD) cgl.o -lfftw3\_mpi -lfftw3 -lm

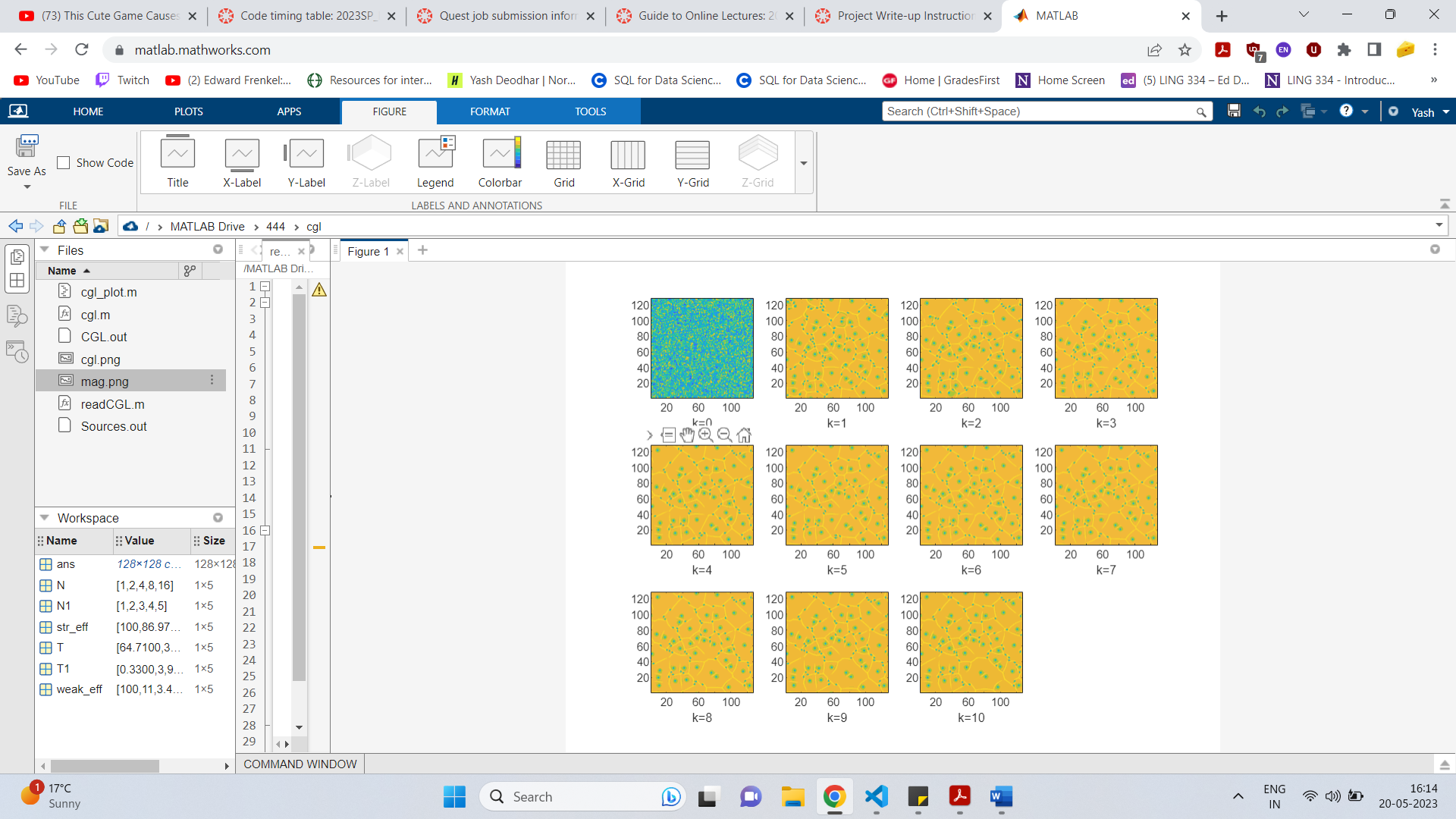
cgl.o: cgl.c

        $(CC) $(CFLAGS) $(ID) -c cgl.c

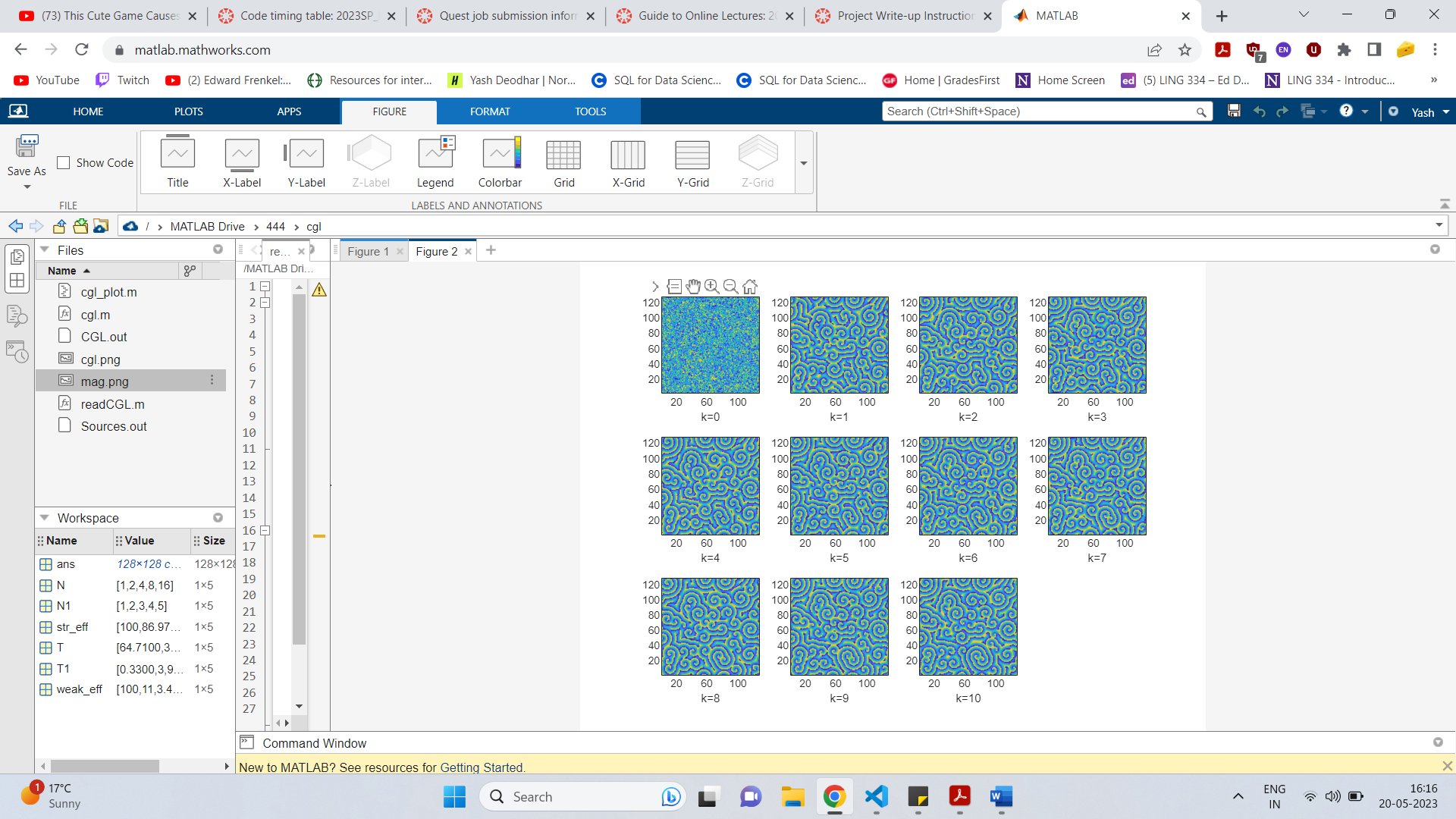
clean:

        rm cgl cgl.o

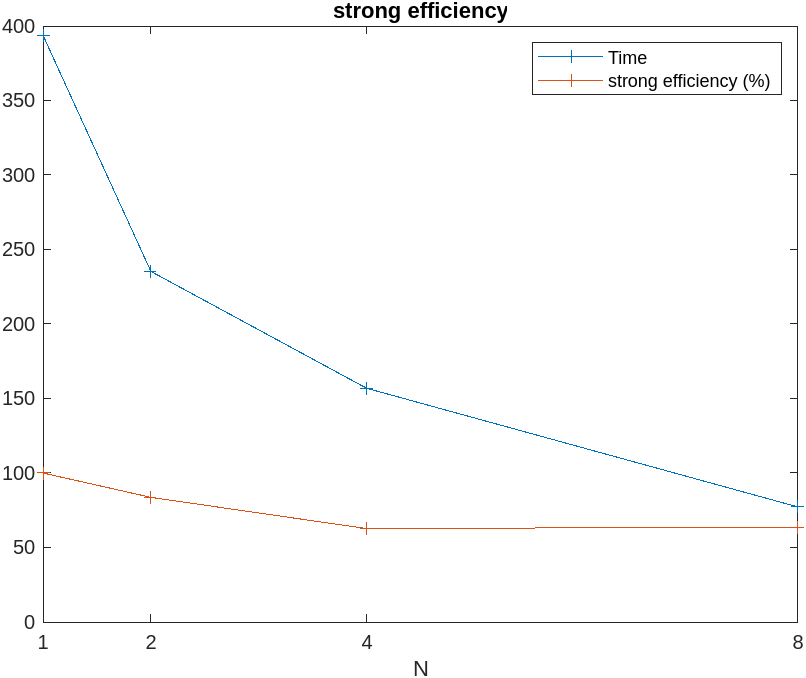
* My plots for N = 128, c1 = 1.5, c3 = 0.25, M = 100000, seed = 1684516224:

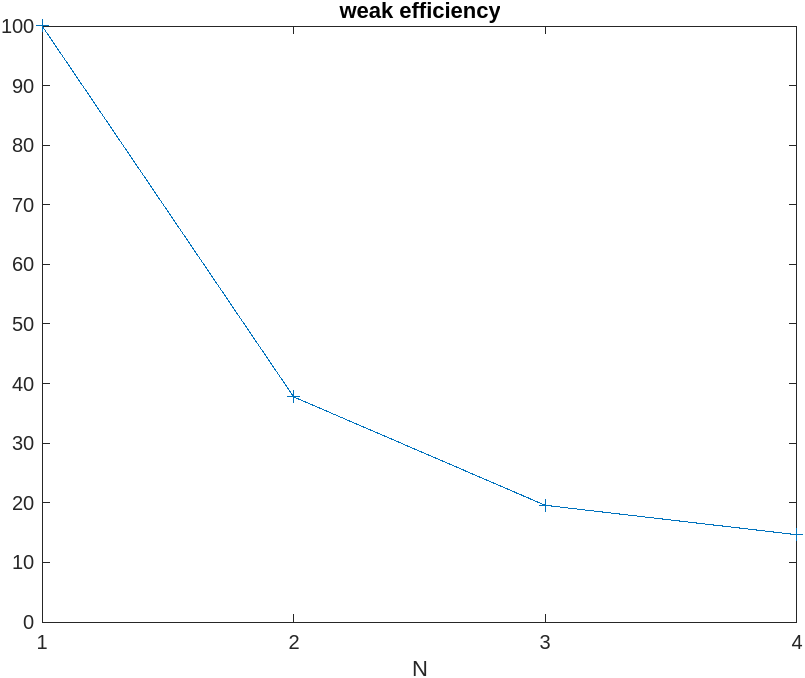
Magnitude of A:

Argument of A:



* Strong efficiency:

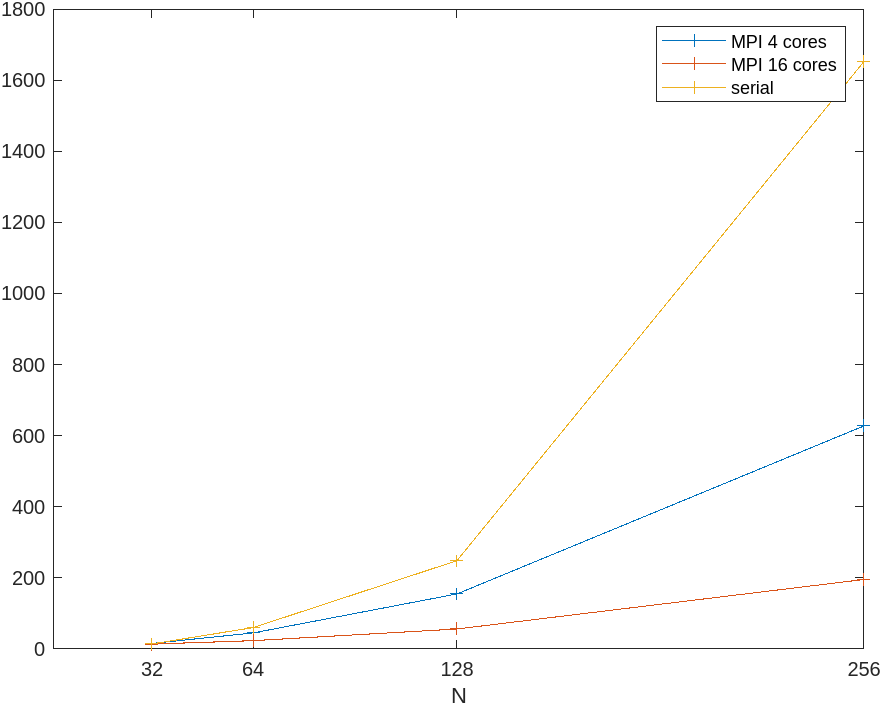


* Weak Efficiency (x-axis values are multiples of N=128):
* Time comparison:

|  |  |  |
| --- | --- | --- |
|  | 4 cores | 16 cores |
| My Time (s) | 156.91 | 91.35 |
| Chopps time (s) | 139.9 | 84.1 |
| Last year’s class avg. (s) | 172.3 | 203.3 |

To make my code faster, I could have tried implementing the FFT method described in the slides instead of using the in-built functions which may have been faster.

* Implementation comparison:



As can be seen from the graph above, the MPI implementation performs better than the serial implementation for all values of N when 16 cores are used. However, when N=32 the serial implementation performs marginally better than the case where 4 cores are used. This suggests that only for very low values of N (N < 32), the serial implementation will perform better than an MPI implantation which uses 4 cores or fewer. This does not seem helpful since for N < 32 no meaningful results will be generated due to such low resolution.

Hence overall, the MPI implementation seems like the better choice.