Modalities

- Example
  - code for matrix multiplication in `/afs/ictp.it/home/o/obrovko/public/num_I/timing`
- Submit
  - by Wednesday, Nov 15, 2017
  - send in the code(s) and the gnuplot script(s)
- Lessons to learn
  - know what operations matter
  - try to think in terms of memory structure (multi-dimensional arrays)
- Optional
  - the part about the sorting algorithm is optional (bonus points)
  - to save data for plotting either write to different files in Fortran or use output redirection in bash
  - if you prefer, you can do the assignment in a programming language of your choice
    - C/C++, Fortran, Python, Java, Perl, PHP, Matlab, R
    - as long as you can install an appropriate compiler on an ICTP machine

come find me f you have question: LB.226 ¦ obrovko@ictp.it

Timing Assignment #1, single numbers

- Preliminary remarks
  - times are very short, so time the execution of 10^6 repetitions of an operations on
  - let us skip the calculation of standard deviation in this part
  - the program then would look something like

```
cpu_time(t1)
do n = 1, reps
   myVar  = 12.345d0
enddo
cpu_time(t2)
print *, 'Time:', (t2-t1) / real(reps) * 1e9
```

  - this shall print the time of one operation in nanoseconds
- Operations to check for integer, real(4) and real(8) :

assignment	`my_var  = 12.345d0`
readout and assignment	`my_var  = my_bar_b`
addition	`my_var  = my_bar_b + my_bar_c`
multiplication	`my_var  = my_bar_b * my_bar_c`

- Post remarks
  - note, that e.g. our multiplication operation consists of reading two values, multiplication and assignment

Timing Assignment #1, 1D and 2D arrays with array operations

Timing of 1D and 2D arrays

- perform timing routine with 10^3 repetitions on 1D and 2D arrays of dimension 1000 and 1000x1000
- do it for double precision only
- print times per array element, i.e.
  1D `time = (t2 - t1) / real(reps) / real(1000) * 1e9`
  2D `time = (t2 - t1) / real(reps) / real(1000000) * 1e9`

Using array operations

- `my_arr` is the 1D array and `my_mat` is the 2D array
- assignment of 1D array
  ```
  my_arr   = 12.345d0
  my_mat   = 12.345d0
  ```
- readout and assignment
  ```
  my_arr   = my_arr_b
  my_mat   = my_mat_b
  ```
- addition
  ```
  my_arr   = my_arr_b + my_arr_c
  my_mat   = my_mat_b + my_arr_c
  ```
- multiplication
  ```
  my_arr   = my_arr_b * my_arr_c
  my_mat   = my_mat_b * my_arr_c
  ```

Timing Assignment #1, 1D and 2D arrays with manual iteration over elements

- Timing of 1D and 2D arrays with manual iteration over elements
  - try to perform the same operations by manually iteration over 1D and 2D array elements with do loops
  - for 2D arrays, try row-major and column-major operations
- Example of manual multiplication
  - row-major (i,j)

```
do i = 1:N
    do j = 1:N
        my_mat(i,j) = my_mat_b(i,j) * my_mat_c(i,j)
    enddo
enddo
```

  - column-major (j,i)

```
do i = 1:N
    do j = 1:N
        array_c(j,i) = my_mat_b(j,i) * my_mat_c(j,i)
    enddo
enddo
```

Timing Assignment #2

- ## Operations' complexity
  - test scaling of manual array operations (assignment, summation, multiplication row-major and column-major)

```
do n=500, 10000, 500    (adjust according to times from assignment #1)
   allocate(matA(n,n))
   allocate(matB(n,n))
   allocate(matC(n,n))
   do rep = 1, 10       (adjust according to times from assignment #1)
      call cpu_time(t1)
      i = 1, n
         j = 1, n
            matC(i,j) = matA(i,j) * mat(i,j)
         enddo
      enddo
      call cpu_time(t2)
      times(rep) = t2-t1
   enddo
   write (outfile,'(I5,F11.3,F11.3)') n, t_mean*1d3, t_stderr*1d3   (time in ms)
enddo
```

  - plot time vs n in gnuplot
- ## Timing sorting routine
  - time the sorting routine for arrays of different lengths (1,10001 in steps of 100)
  - plot time needed to sort vs n
  - randomize array before each sorting