How to get the folders: from GALILEO,
wget --no-check-certificate https://hpc-forge.cineca.it/files/ScuolaCalcoloParallelo_WebDAV/public/anno-2016/25_Summer_School/ex-prof_deb.tar
tar -xvf ex-prof_deb.tar

DEBUGGING & PROFILING EXERCISES

1. **Valgrind**

In the "valgrind" folder there are some small, easy codes with simple errors of memory leaking or uninitialized variables. They compile fine and maybe even execute fine, but they are wrong in the output or waste memory. Use Valgrind to find them and correct them. Launch Valgrind again until it doesn't display any error message.

Load the Valgrind module and then execute Valgrind with the compiled executables (they are serial):
valgrind --tool=memcheck --leak-check=full ./a.out

2. **GDB**

In the "gdb" folder there is the checkprime example from the slides. Try to compile it (both *main.c* and *checkprime.c* in the same line) and experiment with gdb in order to recreate the same debug investigation from the lectures. Note that gdb is provided by default and doesn't need any module to be loaded for it.
The Valgrind exercises can be tested on GDB, as well.
For a quick reference guide about GDB commands:
http://users.ece.utexas.edu/~adnan/gdb-refcard.pdf

3. **Totalview**

In the "totalview" folder there are 3 sub-folders: "C" and "Fortran" contain the different language versions of the same, easy programs with some small error on the MPI functions to detect. MatMul is a program to execute optimized matrix multiplication. Compile it by executing the *compile.sh* script and submit it. You will get an error almost immediately. Use Totalview to debug the code and find out what is the error and in which subroutine of which source file it is located.

To use Totalview on GALILEO:

- Download RCM (Remote Connection Manager) for your Operative System on this page: https://hpc-forge.cineca.it/svn/RemoteGraph/branch/multivnc/build/dist/Releases/
- Create a graphical session with RCM. Login using GALILEO credentials and start a new display
- Modify the jobscript (job.sh – create a new one for the small exercises) and launch a Totalview job following these instructions:
  https://wiki.u-gov.it/confluence/display/SCAIUS/Totalview+with+RCM
  In the case of MatMul, the command line at the end of the job script should become:
  totalview mm_mul.x -a -matsiz $MATSIZ
- On the Startup parameters window, go to the Parallel tab and choose "OpenMPI as parallel system", 1 node and as many tasks as you need for the specific exercise
- Have fun starting and repeating the software execution, adding breakpoints and examining the values of variables until you find the solutions!

4. Scalasca

First of all, remember to login with *ssh –X* for the scalasca GUI to be visible.
For this exercise two versions of a popular molecular dynamics code will be used: **DL_POLY classic** (for replicated data) and **DL_POLY v4** (for domain decomposition). This program was chosen because it is relatively simple to install and run and allows us easily to test directly the main parallelisation strategies. Both applications are available as modules on GALILEO. A Scalasca version of the software is also available (called <executable>.scalasca)

As input we have a chosen a simple system of Lennard-Jones particles which can be used to model simple, atomic liquids (e.g. liquid argon). No electrostatic interactions are included so performance should be entirely due the parallelisation strategy to non-bonded forces.

In the "scalasca" folder you will find the input files (don't move them) and a job script to edit in order to try and execute the scalasca version of DL_POLY. The script must be executed two times: one for DL_POLY classic and the other for DL_POLY_4 (comment and uncomment the job script when necessary).
The scalasca module can be found in profile/base.
A successful run should generate a directory called epik_DLPOLY_O_sum (or similar). Analyze this directory with the examine option of scalasca.
You might like to open two scalasca sessions (one for each program) so you can compare them directly. Scalasca allows you to compare various metrics - some you could look at include:

- Load balancing - Select Computational balance (Metric tree)+System tree
- Bytes transferred. (Metric tree)

Question: How does the load balancing compare between the two cases ?
(i.e. the ratio of the process with the largest load compared to that with the smallest)
Question: Which version transfers more bytes via collective calls and which via point-to-point?
Question: In DL_POLY2, which MPI call transfers the most data?