How to get the folders: from EURORA,
wget --no-check-certificate https://hpc-forge.cineca.it/files/ScuolaCalcoloParallelo_WebDAV/public/anno-2015/24_Summer_School/ex-prof-deb.tar
tar -xvf ex-prof-deb.tar

PROFILING EXERCISES

1. Time

Try using the "time" function with one of the exercises from the MPI Online Training exercise session.
Modify the job script this way:
time mpirun ./executable
And then check for the timing report in your standard error file (#PBS -e).

Try also to add ETIME/SYSTEM_CLOCK routines (if Fortran) or gettimeofday function (if C) to your code as an alternative way to measure the duration of your job.

2. Gprof

In the "gprof" folder you find *transport*, a serial program to evolve a motion equation. Find more info about this at link:
http://www.hpc.cineca.it/content/code-parallelizationhybridization

Compile it with profiling flags and use gprof to analyze the gmon.out with the flat profile and the call graph informations.

Several things to note:

- Every function has a unique index number associated with it to aid in cross-referencing its information elsewhere in the report
- Every function shows it's parent (calling) function and any children (called) functions.
- The routines that consume the most cpu appear at the top of the report
- Several different timing statistics are provided for each function
- The report is actually comprised of three sections in the following order:
    1. Call Graph section
    2. Flat profile (similar to what prof produces)
    3. Index summary

In the "gprof" folder you also find *mpi_prime*, a parallel program to generate prime numbers. It prints out the largest prime number and the total prime numbers up to a specified limit. It requires 4 or 8 MPI tasks to run.

Compile it with profiling flags on and use gprof to analyze the gmon.out with the flat profile and the call graph informations.
Remember to export the GMON_OUT_PREFIX variable in the job script to avoid overwriting on the same gmon.out file:
export GMON_OUT_PREFIX=gmon.out
Various output files will be produced, one for each MPI task. Use gprof with one or more of them. What do you notice? Is there something that must be important but is missing here?

3. **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 EURORA. 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/advanced. Use this version of the module:

module load profile/advanced
module load scalasca/1.4.3_openmpi--1.6.4--intel--cs-xe-2013--binary

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?

DEBUGGING 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.

Valgrind module is in profile/advanced. Load it this way:
module load profile/advanced
module load valgrind

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 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 EURORA:

- 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 EURORA credentials and start a new display
- Modify the jobscript (job.sh) and launch a Totalview job following these instructions: http://www.hpc.cineca.it/content/using-totalview-remote-connection-manager#Eurora
  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 16 tasks
- Have fun starting and repeating the software execution, adding breakpoints and examining the values of variables until you find the solution!