Programming Environment for Practicals

Your course account is: username
Your account password is: password

Accessing CSCS systems - the front end node Ela
You will be given a course account with a username of the form `courseXX` (for some number `XX`) and a password to use for the practical sessions.

In order to gain access to CSCS systems you need to first access our front-end machine Ela which is accessible as `ela.cscs.ch`.

Access Ela by means of ssh as follows:-

```
ssh -Y username@ela.cscs.ch
```

Accessing CSCS systems -Todi
The machine that we will use for all practicals is called Todi, a Cray XK7. Todi is the first GPU/CPU hybrid supercomputing system with high scalability at CSCS, designed to run data parallel and computationally intensive applications.

It features 272 nodes, each one equipped with 16-core AMD Opteron CPU, 32 GB DDR3 memory and one NVIDIA Tesla K20X GPU with 6 GB of GDDR5 memory for a total of 4352 cores and 272 GPUs.

To access Todi:

```
ssh -Y todi
```

Default Environment on Todi
Having logged in to Todi, you will have a default basic environment and directories in 2 file systems which you can access as $HOME and $SCRATCH.

CSCS uses the module command to change your programming environment.
If you issue the command "`module list`" you will see your currently loaded modules.
If you issue the command "`module avail`" you will see all of the available modules that you could load.
If you want to load a module then issue the command "`module load` <modulename>" for some <modulename>, for example "`module load cray-hdf5` ". If you want to swap modules, issue the command "module swap <modulenamecurrent> <modulenamedesired>".
For a simple description of what a module provides use "`module help` <modulename>".

Compilation Environment
In order to compile codes you will need to select a programming environment for a specific compiler. Available compilers on Daint are the Cray compiler (default), Intel, Gnu, and PGI and these are loaded using the module names as shown in the following table.

Compiler	Module Name
Cray	`PrgEnv-cray`
Intel	`PrgEnv-intel`
Gnu	`PrgEnv-gnu`
PGI	`PrgEnv-pgi`

For GPU code:

Modules for CUDA: module load cudatoolkit
Module for OpenACC: module load craype-accel-nvidia35

To compile CUDA code use nvcc

The compiler programming environment provides convenient wrappers that you can use for compilation, and these wrappers ensure that any libraries and header files that you have loaded through the module command are included or linked automatically.

Handy Table for compilation:

	Fortran	C	C++	w/OpenMP	w/OpenACC
CRAY	ftn	cc	CC	by default	-h acc
PGI	ftn	cc	CC	-mp	-acc
INTEL	ftn	cc	CC	-openmp	N/A
GNU	ftn	cc	CC	-fopenmp	announced

Launching jobs

We have a special reservation on the machine "course" which is only available to the course accounts and we need to request nodes from this reservation. For simple practicals 1 node should be sufficient. You should therefore issue the following command which will give you one node up to 1 hour:-

```
salloc --res=course -N 1 --time=01:00:00
```

You will then have your prompt returned after a message such as the following:-

```
salloc: Granted job allocation XXXX
```

You are now able to launch your jobs on the compute nodes.

Running MPI Jobs

For MPI jobs, for example that are to be launched on the compute nodes you need to use the "`-n`" flag to specify how many processes you wish to launch. For example, to launch 8 processes of the myexe1 executable you would issue the following command:-

```
aprun -n 8 ./myexe1
```

Running OpenMP jobs

To launch i.e 8 threads of an OpenMP job you need to specify the number of threads using "export `OMP_NUM_THREADS=8`" and then you need to tell `aprun` that you want 1 process using the "`-n`" flag with 8 threads using the "`-d`" flag as follows:-

```
export OMP_NUM_THREADS=8
aprun -n 1 -d 8 ./ompexe1
```