How to run a parallel job?

How to run a parallel job?

To run parallel processes you need to inform the scheduler that you want to allocate multiple slots (== cpu cores) specifying which operations it needs to do to prepare your parallel environment for your execution.

There are two main types of parallel environments: one that restricts the allocation of the slots within the same compute node, and another more general, that will allow to grab slots from different nodes.

Given the very minimal network infrastructure that we currently have on our system, using inter-nodes communication is not recommended, unless you really know what you are doing.

To allocate \( N \) slots on a single node you need to add the option `'-pe orte N'` to your submission. To allocate \( N \) slots, regardless of where, you have to use the option `'-pe orte-fu N'`.

At runtime the system will define for you the environmental variable `$NSLOTS` that you can use in your script to specify how many processes your parallel execution will have to use.

The system will also define the variable `$PE_HOSTFILE` that points to a file that contains the list of the hosts that your job is running on. This is sometimes needed for some software.

The two main typical APIs used for parallel applications are **MPI** and **OpenMP**.

Running an MPI job:

**MPI** (Message Passing Interface) is a message passing library standard. It has implementations for nearly all Operating systems and network hardware vendors, and it's de-facto the most diffused and supported paradigm people use to design communication among processes in scientific computing in a distributed memory environment.

The implementation that we support on the cluster is **OpenMPI**. To use it, just load the appropriate module (both at compilation time and at runtime in your script).

To execute your application you need to use the command `mpirun` and specify the number of processes that you want to initiate.

```
$ cat simple_mpi_job.txt
#$ -q all.q
#$ -cwd
#$ -N simple_mpi_run
#$ -pe orte 12             ### will request an allocation of 12 slots on a single node.

module load INTEL-2013/OPENMPI/1.7.2_runtime   ### will load the runtime environment for the mpi run

mpirun -np $NSLOTS ./my_parallel_code.x  -in myinput
### the variable $NSLOTS will be set by the scheduler to the number of allotted slot (12 in this case ).
```

Running an OpenMP job:

Since the number of cores and the amount of memory on a single machine can be pretty large these days, it is sometimes an unnecessary complication to go and explicitly design an interprocess communication via message passing, when we can use the shared memory to communicate among processes that live on the same physical node.

**OpenMP** is an API specification for implementing parallelization in a shared memory environment that is very popular and supported by the main compilers.

To run your OpenMP code on a node, depending on what API function you used, you probably need to set an environmental variable that tells the
runtime environment how many processes you want to enable in the parallel sections of your code.

Remember that this should correspond to the number of slots you have allocated in your job submission. Remember also that OpenMP works only in a shared memory environment. So you necessarily need to allocate slots on the same node using `-pe orte N`

```bash
[testuser@login-node-1-0 basic-test]$ cat simple_openmp_job.txt
#$ -q all.q
#$ -cwd
#$ -N simple_OpenMP_run
#$ -pe orte 12             ### will request an allocation of 12 slots on a single node.

cd $SGE_O_WORKDIR

### the variable $NSLOTS will be set by the scheduler to the number of allotted slot (12 in this case).
export OMP_NUM_THREADS=$NSLOTS
./my_openMP_parallel_code.x -in myinput
```