MPI

MPI (or Message Passing Interface) is a standard for passing messages between processes running on the same or different computers. The standard can be found at the MPI Forum website. There are various implementations of the MPI standard, the two main open implementations (as opposed to commercial ones) are:

- OpenMPI
- MPICH

Various versions of both of these packages are installed on the cluster - this is because some of the versions have unfixed bugs but other software depends on those versions.

Usage

All of the MPI packages can be used via the environment modules command:

```
$ module avail --long 2>&1 | grep -i mpi/
mpi/mpich-3.0-x86_64                                  2016/11/04 22:34:42
mpi/mpich-x86_64                                      2016/11/04 22:34:42
mpi/openmpi-x86_64                                    2017/08/03  8:05:05
```

The openmpi-x86_64 and mpich-x86_64 versions are those that come with the operating system. The other versions have been compiled locally.

mpich-x86_64 doesn't support InfiniBand (the high-speed network that links the cluster nodes) - we therefore recommend that you use mpich/3 or later or openmpi.

The following example will show how to run an MPI program using openmpi-x86_64. First we need an MPI program, we will use an example written in C:

```c
// module load mpi/openmpi-x86_64
// mpicc -g mpi-test.c -o mpi-test
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <unistd.h>

int main (int argc, char **argv) {
    int rank, size, hostlen;
    char hostname[MPI_MAX_PROCESSOR_NAME];

    // Deliberately ignore errors
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Get_processor_name(hostname, &hostlen);
    printf("I am %i of %i on %s\n", rank, size, hostname);
    MPI_Finalize();
```

```
    return EXIT_SUCCESS;
}
```

The program starts the MPI environment, finds how many processes there are in total, then each process prints it's position and the name of the host it's running on. The program is compiled with the commands:

```
$ module load mpi/openmpi-x86_64
$ mpicc -g mpi-test.c -o mpi-test
```

and can then be run with the following submit script:

```
#!/bin/bash
module load mpi/openmpi-x86_64
mpiexec --prefix $MPI_HOME --hostfile $PBS_NODEFILE --map-by
ppr:$PBS_NUM_PPN:node ./mpi-test
```

which can be submitted with the command:

```
$ qsub -l nodes=10:ppn=2 mpi-test.sh
```

This will run the job on 10 hosts, with 2 processes on each node. The output of the job looks like this:

```
I am 0 of 20 on ava04.grid.fe.up.pt
I am 1 of 20 on ava04.grid.fe.up.pt
I am 12 of 20 on ava12.grid.fe.up.pt
I am 13 of 20 on ava12.grid.fe.up.pt
I am 2 of 20 on ava14.grid.fe.up.pt
I am 3 of 20 on ava14.grid.fe.up.pt
I am 10 of 20 on ava18.grid.fe.up.pt
I am 11 of 20 on ava18.grid.fe.up.pt
I am 18 of 20 on cfp06.grid.fe.up.pt
I am 19 of 20 on cfp06.grid.fe.up.pt
I am 4 of 20 on ava17.grid.fe.up.pt
I am 5 of 20 on ava17.grid.fe.up.pt
I am 8 of 20 on ava19.grid.fe.up.pt
I am 9 of 20 on ava19.grid.fe.up.pt
I am 16 of 20 on ava02.grid.fe.up.pt
I am 17 of 20 on ava02.grid.fe.up.pt
I am 14 of 20 on ava20.grid.fe.up.pt
I am 15 of 20 on ava20.grid.fe.up.pt
I am 6 of 20 on ava10.grid.fe.up.pt
I am 7 of 20 on ava10.grid.fe.up.pt
```

mpiexec

There are a couple of important things about the mpiexec command we used in our submit script above:

```
mpiexec --prefix $MPI_HOME --hostfile $PBS_NODEFILE --map-by
ppr:$PBS_NUM_PPN:node ./mpi-test
```

The first is the *–prefix $MPI_HOME* argument. This is needed for the *mpiexec* startup process - the command is run on one node first, and then starts itself on the other nodes, without the *–prefix* argument it doesn't know where to find itself and it will fail.

The second is the *–hostfile $PBS_NODEFILE* argument. This argument tells MPI to read the file *$PBS_NODEFILE* to find out which nodes it should run on. This file is automatically generated by the queuing system, so all you need to do is to include the reference and it will start using it.

Finally the *–map-by ppr:$PBS_NUM_PPN:node* argument tells MPI how many processes to spawn on each node.

A final important option is one we didn't need above. This is the -x option. This exports the environment variable given as it's argument to the MPI program - if it's not included, then the MPI programs will not inherit the variable. It's especially useful if you are using a scripting language such as python and it's MPI bindings module mpi4py. If you don't do:

```
module load mpi/openmpi-x86_64
mpiexec --prefix $MPI_HOME --hostfile $PBS_NODEFILE --map-by
ppr:$PBS_NUM_PPN:node -x PYTHONPATH ./mpi-test.py
```

then the python program won't be able to find the MPI bindings and it will fail to run.