Solution 1

Solution 1

C

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>

int main(int argc, char* argv[]){

  int me, nprocs;

  /* Initialize MPI environment */
  MPI_Init(&argc, &argv) ;

  /* Get the size of the MPI_COMM_WORLD communicator */
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs) ;

  /* Get my rank... */
  MPI_Comm_rank(MPI_COMM_WORLD, &me) ;

  /* ...and print it. */
  printf("\tHello, I am task %d of %d.\n", me, nprocs);
  /* Finalize MPI environment */
  MPI_Finalize() ;

  return 0;
}
```

FORTRAN

```
 program hello
```

```
  !$ Use MPI module in Fortran 90
  use mpi
  implicit none

  integer ierr,me,nprocs

  !$ Initialize MPI environment
  call MPI_INIT(ierr)
  !$ Get the size of the MPI_COMM_WORLD communicator
  call MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
  !$ Get my rank... (zero-based even if we are in Fortran)
  call MPI_COMM_RANK(MPI_COMM_WORLD, me, ierr)
  !$ ...and print it.
  print *,"Hello, I am task ",me," of ",nprocs,"."
  !$  Finalize MPI environment
  call MPI_FINALIZE(ierr)

end program hello
```

COMPILING & EXECUTING ON FERMI

For **compiling** your code, remember that you have to cross-compile, since your program will run on the differently structured back-end nodes. In order to do so, you have to load one of the proper compiler modules:

```
module load bgq-xl (recommended)
module load bgq-gnu
```

With the command "module help bgq-xl (or bgq-gnu)" you can see what compiler you have to use for your specific programming language. After compilation, you are ready to execute your job.

To submit the execution as a **batch job** try with:

```
#!/bin/bash
# @ job_name = exercise_1
# @ output = ex1.out
# @ error = ex1.err
# @ shell = /bin/bash
# @ job_type = bluegene
# @ wall_clock_limit = 00:30:00
# @ bg_size = 128
# @ account_no = <account>
# @ class = training
# @ queue

runjob --np 4 --exe ./ex1.x
```

Please note:

- You have to insert your account name in the "account_no" field. Type:

```
saldo -b
```

to know what your account name is.

- During the school, you can launch the job run in a special high-priority class called "training", but you have to specify it in the jobscript (with the LL keyword "class = training"). You also need to specify a job size of 128: class training won't work with 64. After the school, you will be able to submit only as a regular user, so you have to remove the "class = training" parameter.

- The parameter --np 4 will make you use only 4 processors of the 2048 (=128x16) you allocated. You may want to try different np values to see how many processors you can involve. No parameter means that all of the 2048 processors will be used.

- Please note the "./" before the executable, otherwise runjob won't find where your executable is.

COMPILING & EXECUTING ON PLX

You can **compile** your code with one of the "openmpi" modules installed on PLX. With:

```
module avail openmpi
```

you get a list of the openmpi modules available (depending on the compiler each module is associated with). You can load the module you prefer with the command module load, for example:

```
module load autoload openmpi/1.4.4--gnu--4.5.2
```

(the autoload option is for loading automatically every module that openmpi may need). After you've compiled your code (mpif90 for Fortran, mpicc for C, mpiCC for C++), you are ready to test it.

To submit the execution as a **batch job** try with:

```
#!/bin/bash
#
#PBS -N myjob
#PBS -q <queue used>
#PBS -o job.out
#PBS -e job.err
#PBS -A <account_no>
#PBS -l walltime=10:00
#PBS -l select=1:ncpus=4:mpiprocs=4

cd $PBS_O_WORKDIR

module load autoload <openmpi module used>
mpirun ./ex1.x
```

You have to insert your account name in the "account_no" field. Type:

```
saldo -b
```

to know what your account name is.

‹ Exercise 1 up Exercise 2 ›