Workshop 6: Introduction to Isca

This workshop is an introduction to using the University's HPC system (called Isca). In many ways the Isca environment is similar to the Linux environment on the Lovelace lab PCs. However one important difference is that on Isca (like other HPC systems) there is a queuing system which allocates resources to jobs. In this workshop you will see how to build and run OpenMP, MPI and hybrid (OpenMP and MPI) programs on Isca. You will also see how to use environment modules to make software packages available and gain experience of using the queuing system.

1 Getting set up

Isca is physically located in the university data centre so it is accessed remotely by connecting to a log in node using SSH. This is very similar to accessing the Lovelace lab PCs using SSH. On the module ELE page there are instructions describing how to log in to a remote Linux computer under the "Workshop 2" section.

- If you are connecting using SSH then open a terminal (Linux/MacOS) or command prompt/power shell window (Windows) and log in by running the command

  ```
  ssh abc123@login.isca.ex.ac.uk
  ```

  where you should replace `abc123` with your university username. You will be prompted for your password, which is the same password used for the Lovelace lab PCs and other University IT systems. If you are connecting using PuTTY you should follow the instructions in the "Remote access using PuTTY" document instead.

  When you have logged in you should see the welcome message and be presented with a command prompt on the log in node:

  ```
  ********************************************************************************
  *                  UNIVERSITY OF EXETER ISCA HPC SERVICE                       *
  *                                                                              *
  * This computer system is operated on behalf of the University of Exeter.      *
  * Only authorised users are entitled to connect and/or login to this           *
  * computer system. If you are not sure whether you are authorised, then        *
  * you are not and should DISCONNECT IMMEDIATELY.                               *
  *                                                                              *
  * For user documentation please visit                                          *
  *    https://universityofexeteruk.sharepoint.com/sites/ExeterARC               *
  * Email isca-support@exeter.ac.uk with any direct support issues               *
  *    *
  *               CURRENT TIMEOUT FOR INACTIVITY IS 24 HOURS                     *
  *    *
  ********************************************************************************
  [abc123@login01 ~]$
  ```

- The example programs for this workshop are provided as a tar file. There are two versions of the tar file (one for ECM3446 and one for ECMM461) Take a copy of the tar file with the correct module code by running either

```
cp /lustre/projects/Research_Project-IscaTraining/HPC/workshop6_ecm3446.tar .
```

or

```
cp /lustre/projects/Research_Project-IscaTraining/HPC/workshop6_ecmm461.tar .
```

where the dot at the end of the command causes the file to be copied into the current working directory. When you first log in to Isca the current working directory is your home directory, which is mounted on the log in nodes and all the compute nodes. There should be sufficient space in your home directory for all the workshops on this module.

- Unpack the tar file by running

```
tar xvf workshop6_ecm3446.tar
```

or

```
tar xvf workshop6_ecmm461.tar
```

This will create a directory called `workshop6` which contains the example programs for this workshop.

2 Compiling and running an OpenMP program

In this section we will compile and run the OpenMP Hello World program as an example of how to build and run an OpenMP program on Isca.

2.1 Compilers and Modules

In order to compile the `hello.c` program we need a C compiler. We shall start by seeing whether the GNU C compiler (`gcc`) is available and if so which version it is.

- Run the command

```
which gcc
```

which tells us whether gcc is available and if so where the executable is to be found. It should report that `gcc` is `/usr/bin/gcc`.

- Next check which version of the `gcc` compiler this is by running

```
gcc --version
```

You should find that `gcc` is version 4.8.5.

The version of GCC in `/usr/bin` is the version distributed with the operating system. `/usr/bin` is a different directory on the log in nodes and on the compute nodes (they have different root filesystems) and there is no guarantee that these will be the same version of the compiler.

Environment modules [1] provide a way to easily switch between different software packages and are often used in HPC environments. We can load an environment module to set up our environment to use a newer compiler version which is consistent between the compute nodes and the log in nodes. For this workshop we will load a module which sets up the "Intel Cluster Toolkit". The Intel Cluster Toolkit includes compilers, an MPI library and an optimised maths library.

- To see whether any modules are currently loaded run the command

[1] `http://modules.sourceforge.net`

```
module list
```

The command should report

```
No Modulefiles Currently Loaded.
```

- You can see which modules are available by running the command `module av` however this will return a long list of all the available modules and it can be difficult to find the modules of interest. Instead run the command

```
module av intel
```

to see which Intel Cluster Toolkit modules are available. The output from this command should show

```
---------------- /gpfs/ts0/shared/modules/all ----------------
intel/2015b       intel/2017a       intelcuda/2016.10
intel/2016b       intel/2017b
```

- Now run the command

```
module load intel/2017b
```

to load the most recent version of the `intel` module. To see what has been loaded run `module list` again and you should see

```
Currently Loaded Modulefiles:
  1) GCCcore/6.4.0
  2) binutils/2.28-GCCcore-6.4.0
  3) icc/2017.4.196-GCC-6.4.0-2.28
  4) ifort/2017.4.196-GCC-6.4.0-2.28
  5) iccifort/2017.4.196-GCC-6.4.0-2.28
  6) impi/2017.3.196-iccifort-2017.4.196-GCC-6.4.0-2.28
  7) iimpi/2017b
  8) imkl/2017.3.196-iimpi-2017b
  9) intel/2017b
```

The `intel/2017b` has been loaded, along with a number of dependencies including an updated GCC version.

- If you run `gcc --version` again you should see that version 6.4.0 of the compiler is now being used and if you run `which gcc` it should report that the path to `gcc` is

```
/gpfs/ts0/shared/software/GCCcore/6.4.0/bin/gcc
```

This path is on a filesystem which is mounted on the compute nodes and the log in nodes so loading this module ensures that a consistent `gcc` version is used to compile and run the program.

- You can now build an executable file from the source code by running the commands

```
cd workshop6
gcc -o hello -fopenmp hello.c
```

where the flag `-fopenmp` enables OpenMP. This will compile the source code in the file `hello.c` and generate an executable file called `hello`.

2.2 Running a job

The workload on an HPC cluster is managed using a queuing system which allocates resources to jobs. There are a number of different HPC queuing systems available but they all work in a similar way. Jobs are submitted to the queue using a script file which requests the resources required to run the job (i.e. a certain number of compute nodes for a certain length of time) and says how to run the job. The queuing system on Isca is called SLURM[2] which is a widely used queuing system in HPC.

The file `run_hello.sh` is a job script which we will use to run the `hello` executable. The job script is a shell script but near the top of the file are a number of lines beginning with the characters `#SBATCH` which are directives for the queuing system. When the job runs the job script is executed as a bash shell script and lines beginning with a `#` (including the `#SBATCH` directives) are treated as comments and ignored.

- The example job script requests one compute node to run one task using 8 processors for a duration of one minute. Read through the job script and identify the directives which give these instructions (the job script is commented to help with this).
- Next submit your job to the queue using the command

  ```
  sbatch run_hello.sh
  ```

  This will return a message which tells you the job number, for example

  ```
  Submitted batch job 474512
  ```

- To see the current status of your job run the command

  ```
  squeue --user=abc123
  ```

  where you should replace `abc123` with your username. The `ST` column is the "job state code" which will be `R` if the job is running and `PD` if the job is waiting to run ("Pending"). If your job has already completed then it will not show up in the output from `squeue`.
- If you need to cancel a job use the `scancel` command (for example to cancel job `474512` run the command `scancel 474512`).
- The output from your job will appear in the file `slurm-123456.out` (where `123456` is the job number). Look in the output file and check that you see the "Hello World" message printed 8 times (once from each OpenMP thread).

2.3 Changing the number of OpenMP threads

We can use SLURM to control the number of OpenMP threads by modifying the `--cpus-per-task` directive in the job script. You will need to open the job script in a text editor and there are a number of text editors available on Isca (including `nano`, `emacs` and `vi`). If you are not familiar with Linux text editors then you may like to use `nano`. If you get stuck in `vi` and have troubling exiting type `:q!` to exit and discard changes.

- Open the job file (`run_hello.sh`) in a text editor and edit the job script to change the number of OpenMP threads from 8 to 16.
- Re-run the job to confirm that the number of "Hello World" messages changes as expected.

Question 1: How did you modify the job script to change the number of OpenMP threads from 8 to 16?

[2] `https://slurm.schedmd.com`

3 Running an MPI program

In this section we will see how to build and run an MPI version of the Hello World program. In this version of the program each process reports its rank and the total number of MPI processes. The program also makes a call to an MPI function called `MPI_Get_processor_name` which returns the name of the compute node where the calling process is running.

- Firstly compile the MPI version of the Hello World program.

  ```
  mpicc -o hello_mpi hello_mpi.c
  ```

- Then run the program by submitting the job script.

  ```
  sbatch run_hello_mpi.sh
  ```

Question 2: How many MPI processes are started by the `mpirun` command and where do they run?

Question 3: How would you modify the job script to run 32 MPI processes on 2 nodes with 16 processes on each node?

4 Running a hybrid program

In this section we will see how to build and run a hybrid (MPI/OpenMP) of the Hello World program.

- Firstly compile the hybrid version of the Hello World program.

  ```
  mpicc -fopenmp -o hello_hybrid hello_hybrid.c
  ```

- Then run the program by submitting the job script.

  ```
  sbatch run_hello_hybrid.sh
  ```

Question 4: How many MPI processes and OpenMP threads are there and where do they run?

Question 5: How would you modify the job script to run 8 MPI processes per node and 2 OpenMP threads per MPI process?