Getting Started on MPI programming

A set of sample MPI programs are provided. Two of these `first` and `second` show how to compute the sum of integers up to a given value. The idea here is to divide the range into equal divisions according to the number of processors. Each process then works on its own partial sum. Finally, all the partial sums are combined to give the full sum. The example programs here illustrate the "master" and "slave" approach. The first example uses just two processors—a master and a slave. The second example uses any number of processors—a single master and several slaves.

To get started, do the following:

1. Login at a terminal in the Computer Vision Lab. Your login name is your student ID prefixed with a lower case `s`. The initial password is your student ID prefixed with an upper case `S`. For example, if your student ID is `12345678`, then your username to enter at the terminal will be `s12345678` and your password is `S12345678`.

2. Bring up a terminal window and log into `mighty` using ssh. As above your username is `s+` your student ID, and your password is `S+` your student ID.

   ```
   ssh -X s12345678@mighty
   ```

   Note the use of the `-X` switch, where the `X` is in upper case

3. Get a copy of the MPI examples from my public repository

   ```
   hg clone https://bitbucket.org/iabond/mpi_examples
   ```

4. Go to the directory with these examples

   ```
   cd mpi_examples
   ```

5. Open either the C program source file `first.c` or the C++ source file `first.cpp` using your favourite editor, eg

   ```
   emacs first.c &
   ```

6. Have a look at the source code and try and figure out what is going on

7. Compile the program

   ```
   mpicc first.c -o first
   ```

   or

   ```
   mpic++ first.cpp -o first
   ```

8. Open the corresponding PBS in an editor.

   `emacs first.pbs &`

9. This file provides specification and operating parameters for when the job is submitted to the cluster. Figure out what is going on here and make appropriate edits to this file so that it specifies *your* program

10. Submit the job to the cluster

    `qsub first.pbs`

11. Use `qstat` to see the status of your job. When the job is finished, examine the output in the destination file specified in `first.pbs`

12. Repeat the above steps 5–11 with `second.c`

Submitting jobs to the cluster

Five computing nodes in the cluster have been reserved for this class. An associated job queue, `pp`, has been created that will allow you to use them. You can submit to this queue by adding `-q pp` to the top line of your pbs file. You can request resources of up to 5 nodes with 8 processors per node. I would recommend using the `-np` option on `mpiexec` to specify the number of parallel processes to create. So your pbs file should look something like

```
#PBS -j oe -o pi.stdout -l nodes=5:ppn=8 -q pp
mpiexec -np 20 -machinefile $PBS_NODEFILE /home/userid/path/to/myprog
```

Checking jobs on the queue

Use `qstat` to check the status of your submission. A job state of `Q` means it is waiting in the queue to run, `R` means that your program is now running on the processors, and `E` means exiting (not error). If another user has a job currently in state `R`, your job will go into state `Q` but you should not have to wait long for your job to start.

The default `inms` queue is now being used very heavily by others in the institute. If you submit any jobs to this queue they may stay in the wait `Q` state for a long time without being run

Deleting jobs

If your job stays in the `R` state for a long time, there is a good chance there is something wrong with it. A properly working implementation of the first assignment should not take more than a minute. If this is the case, please use `qdel` followed by the job number to free up the processors for others.