OSSCOM

Tempus

Partnership with Enterprises Towards Building Open Source Software Communities and Rejuvenation of Technical Education and Innovation

Grant agreement no.: No 544520-TEMPUS-1-2013-1-DE-TEMPUS-JPHES

Parallel programming
Lab 0: Introduction to the lab environment

Objectives

The objectives of this introductory lab are:

- Learning how to compile, link and run parallel programs that use OpenMP, PThreads and/or MPI.
- Using the TORQUE resource manager

1 Connecting to the Linux cluster

Access to the Linux cluster is through secure shell (ssh) to 161.67.132.7. From Linux you should type `user@161.67.132.7`, where *user* is your login user in the main node of the cluster. From Windows you can use PuTTY, a free implementation of telnet and ssh.

The cluster consists of 7 machines, 1 of them is the main (access or login) node and 6 are slave or compute nodes. All of them share HOME directories via NFS. The 7 nodes belong to the *HP ProLiant DL370 G6 Server* series. Every node has:

- 2 Intel® Xeon® Processors E5506 Quad core (4M Cache, 2.13 GHz, 4.80 GT/s Intel® QPI)
- Main memory: 12 GB
- Hard disk: the main node has 1 TB (shared with slave nodes via NFS) plus 250 GB (boot). The slave nodes have 500 GB. Everything is mirrored for security purposes.
- Network: 3 Gigabit Ethernet adapters
- Graphics card: NVIDIA Quadro FX5800 4GB

This is a list of installed software related to parallel programming:

- GCC compiler with support for OpenMP (version 3.0) and PThreads
- MPICH2, a high-performance and widely portable implementation of the Message Passing Interface (MPI) standard (both MPI-1 and MPI-2)
- TORQUE (Terascale Open-Source Resource and QUEue Manager), an open source distributed resource manager providing control over batch jobs and distributed compute nodes. TORQUE is a community project based on the original PBS (Portable Batch System) project.

2 Compiling and linking parallel programs

MPI	`mpicc mpi_foo -o mpi_foo`
PThreads	`gcc -pthread pthread_foo.c -o pthread_foo`
OpenMP	`gcc -fopenmp omp_foo.c -o omp_foo`

While for compiling OpenMP and PThreads sources we just invoke `gcc`, in order to compile MPI programs is recommended to use a wrapper, `mpicc`.

This wrapper do not do any compiling or linking, it just manipulates the command line, adds relevant compiler/linker flags and then invokes the real compiler. You can run

```
mpi -compile_info
mpi -link_info
```

to get these flags and options.

Exercise 1. Download `sources_lab0.tar.gz` from moodle and compile the 3 source files in order to get the executables.

3 Running locally parallel programs

In order to execute OpenMP and PThreads programs we just need to run the executable file. The number of created threads will be given by the program itself or environment variables (this will be seen in detail in labs 3 and 4).

MPI programs need more information to run. We need to use the order `mpiexec` and, at least, we need to say how many processes will be created (default is 1). In this example

```
mpiexec -n 4 ./mpi_foo
```

the `-n` option specifies the number of processes to run on the machine. In this case 4 processes are running `mpi_foo`

In order to test locally MPI programs, a config file and a deamon are needed. First it is necessary to launch the *mpd daemon* (multi-purpose daemon). You can start an mpd on the main node with `mpd &`. Once we have the *mpd daemon* running, you need to create the config file. You can follow these instructions to create the file `.mpd.conf` in your HOME directory:

```
A file named .mpd.conf file must be present in the user's home
directory (/etc/mpd.conf if root) with read and write access
only for the user, and must contain at least a line with:
MPD_SECRETWORD=<secretword>
One way to safely create this file is to do the following:
  cd $HOME
  touch .mpd.conf
  chmod 600 .mpd.conf
and then use an editor to insert a line like
  MPD_SECRETWORD=mr45-j9z
into the file.  (Of course use some other secret word than mr45-j9z.)
```

Exercise 2. Run locally (in the main node of the cluster) the 3 executable files you got in Exercise 1. Run the MPI program for 1, 4 and 8 processors.

4 Running parallel programs in a cluster throught TORQUE

A job in TORQUE has to follow these stages: (1) creation, (2) submission, (3) execution, and (4) finalization.

- Creation. First, we need to write a *job script* that contains the *parameters* of a job, e.g., how long a job should run (walltime), what resources are necessary to run, and what to execute. Job scripts follow PBS syntax.
- Submission. Once we have the script we use `qsub jobscript` to submit the job.
- Execution. While a job is running, its status can be queried with `qstat`.
- Finalization. When a job completes the stdout and stderr files are copied to the directory where the job was submitted. We can also abort the execution of a job using `qdel jobID`

Some of the most commonly used PBS options and variables in a job script file are:

Option	Description
#PBS -N myJob	Assigns a job name. The default is the name of PBS job script.
#PBS -l nodes=4:ppn=2	The number of nodes (nodes) and processors per node (ppn).
#PBS -q queuename	Assigns the queue your job will use.
#PBS -l walltime=01:00:00	The maximum wall-clock time during which this job can run.
#PBS -o mypath/my.out	The path and file name for standard output.
#PBS -e mypath/my.err	The path and file name for standard error.
#PBS -j oe	Join option that merges the standard error stream with the standard output stream of the job.
#PBS -m b	Sends mail to the user when the job begins.
#PBS -m e	Sends mail to the user when the job ends.
#PBS -m a	Sends mail to the user when job aborts (with an error).
#PBS -m ba	Allows a user to have more than one command with the same flag by grouping the messages together on one line, else only the last command gets executed.
#PBS -r n	Indicates that a job should not rerun if it fails.
#PBS -V	Exports all environment variables to the job.

Variable	Description
PBS_JOBID	Job identifier
PBS_O_WORKDIR	Working directory

This is an example of job script for an OpenMP program (file `script_omp.job`):

```
#!/bin/bash
#PBS -N script_omp
#PBS -o script_omp.$PBS_JOBID.out
#PBS -e script_omp.$PBS_JOBID.err
#PBS -l nodes=1:ppn=8
#PBS -l walltime=00:03:00
cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=8        ##number of threads
./omp_foo
```

And this is an example of an MPI job script (file `script_mpi.job`):

```
#!/bin/bash
#PBS -N script_mpi
#PBS -o script_mpi.$PBS_JOBID.out
#PBS -e script_mpi.$PBS_JOBID.err
#PBS -l nodes=2:ppn=8
#PBS -l walltime=00:30:00
NP=$(wc -l $PBS_NODEFILE | awk '{print $1}')
cd $PBS_O_WORKDIR
mpiexec -n $NP -machinefile $PBS_NODEFILE ./mpi_foo
```

More information about TORQUE in: http://www.clusterresources.com/products/torque/docs/ (chapter 2)

Exercise 3. Run through TORQUE the 3 programs in previous exercises. Run OpenMP programs for 4 and 8 threads, and the MPI program for 16 and 24 processors.