An Introduction to HPC and Scientific Computing

Lecture three: Introduction to Linux, compilers and build systems

Ian Bush

Oxford e-Research Centre,
Department of Engineering Science

Overview

In this **short** lecture we will learn about:

- Linux
- Compilers and make
- A little bit about Arcus-b – the machine that we will do the practicals on
- Batch Systems

Linux

- Linux is the operating system (OS) that the vast majority of supercomputers use
  - Derived ultimately from Unix
- For us, that controls how we interact with the machine
  - You are probably more used to Microsoft Windows or OS X on a Mac
- Windows and Macs tend to stress the graphical user interface
  - But note under the hood OS X is also derived from Unix
- While Linux does have an extensive GUI most people work at the command line
  - So instead of pointing at and clicking things you type commands in a terminal at a prompt

Logging onto Arcus-b

- You log onto a remote system using the ssh command
- We will do the practicals on arcus-b, the University's central compute cluster run by ARC
- Here I log onto arcus-b and show the `ls` command which lists the files in my home directory
  - Note here I am using mobaxterm - the software I recommend for use on Windows systems

```
[ijb.DESKTOP-66DFAKH] ➤ ssh -CX oerc0085@arcus-b.arc.ox.ac.uk
oerc0085@arcus-b.arc.ox.ac.uk's password:
Last login: Mon May 14 13:09:45 2018 from oerc-dynamic-185.oerc.ox.ac.uk
*********************************************************************
* Welcome to arcus-b - a Haswell IBM/Lenovo NeXtScale cluster      *
*                                                                   *
* Please report all issues to support@arc.ox.ac.uk                  *
*                                                                   *
* /home/group/name (or $HOME) = Your (small) home area              *
* /data/group/name (or $DATA) = Your working area for data storage *
*********************************************************************
[oerc0085@login11(arcus-b) ~]$ ls
arcus.inc  barry.tar.gz  bin  CRYSTAL  FoldersDeleted-20170816  FoldersTargettedForDeletion-20170714  intel  script.sl  Wes
[oerc0085@login11(arcus-b) ~]$
```

Computing – You only learn by doing it

- You only really learn this stuff by doing it

- Hence rather than go through all of numerous command in Linux there is a worksheet which will be part of the practical where you will learn what you need

- However it's worth noting that the Unix, and hence Linux, philosophy is a bit different from that found in Windows

- In Unix you have a large number of relatively simple tools which can easily be chained together to do something quite complicated
  - This is flexible and can do many things, but can take a while to learn

- Under Windows you tend to have one big tool that does complicated things
  - Great and easy to use if you want to do what it can do
  - But not great once you stray outside those bounds

- Anyway, we'll learn Linux by doing it!

Using Compilers

- As you learnt earlier a compiler converts a High-level language such as C into the machine code that the computer understands

- Let’s look a little more at that and how it works under Linux

HLL

```
int square(int num) {

    return num * num;

}
```

Assembly Code

```
push rbp #2.21
mov rbp, rsp #2.21
sub rsp, 16 #2.21
mov DWORD PTR [-16+rbp], edi #2.21
mov eax, DWORD PTR [-16+rbp] #3.18
imul eax, DWORD PTR [-16+rbp] #3.18
leave #3.18
ret #3.18
```

Machine Code

```
01110000011010001010100100100101001
00011111111111010101111001110010 10
10101010011111111111010101001010 10
10101010101010101001010101010100101
```

https://godbolt.org/

There is No One True Compiler

- There are many compilers available under Linux

- The most commonly used are
  - gcc – The Gnu Compiler
    - Truly free, comes with every Linux
  - icc – The Intel Compiler
    - Commercial (i.e. costs money), not always available, but generally produces executables that run faster than those from the Gnu compiler

Using A Compiler Under Linux

```
> ls
hello.c
> cat hello.c
#include <stdio.h>
#include <stdlib.h>
int main( void ){
  printf( "Hello World!\n" );
  return EXIT_SUCCESS;
}
> gcc hello.c
> ls
a.out  hello.c
> ./a.out
Hello World!
>
```

- Note
  - You should always end your C file names with `.c` – confusion will occur if you do not!
  - The default name for an executable under Linux is a.out
  - You run an executable by using its name

Compilers with Flags

```
> ls
hello.c
> cat hello.c
#include <stdio.h>
#include <stdlib.h>
int main( void ){
  printf( "Hello World!\n" );
  return EXIT_SUCCESS;
}
> gcc hello.c -o hello
> ls
hello  hello.c
> ./hello
Hello World!
> 
```

- Compilers can take many flags which alter how they behave
  - Here we make the executable have a more sensible name
  - In the practical we will see more examples of this

Programs in Multiple Files

- Note a program need not be all in one file:

```
> ls
print_squares.c  square.c
> cat print_squares.c
#include <stdlib.h>
#include <stdio.h>

int square( int );

int main( void ){

  int n = 3;

  printf( "Square of %d is %d\n", n, square( n ) );

}
> cat square.c
int square( int num ) {
  return num * num;
}
> gcc print_squares.c square.c
> ./a.out
Square of 3 is 9
> 
```

Compiling and Linking

- Note there are actually two phases in this process
- We first compile each file to a corresponding *object file*
  - Thus each source file has a corresponding object file
- Then all the object files are *linked* together to produce the executable
- By default both are done
- The `-c` flag can be used to force only the compilation stage

```
> ls
print_squares.c  square.c
> gcc -c print_squares.c
> ls
print_squares.c  print_squares.o  square.c
> gcc -c square.c
> ls
print_squares.c  print_squares.o  square.c  square.o
> gcc print_squares.o square.o -o print_squares
> ./print_squares
Square of 3 is 9
>
```

Compiling and Linking

- Why is this useful?
- Let’s pretend we have a program in 100 different files
- We now compile all those 100 files and link the resulting object files together to produce an executable
- We now change just one of the source files
- Without the 2 stage system we would have to recompile everything
- But now we can just compile the one source file that has chaned to a new object file, and then relink that with the existing 99 old ones to produce a new executable
- This can save a lot of time in a big program!
- It is so useful there is a special utility called `make` that takes advantage of this
  - We don’t need to know much about make as we will provide the makefiles for all the exercises
  - We mainly need to know how to use it

Using make

- Note how the Makefile contains a set of rules that `make` interprets
  - Don't worry, we will always provide this!
- Note how it also automatically works out what files need to be compiled and only compiles them
- Also `make clean` is very common – clean up and leave the files as they were before any compilation occurred

```
> ls
Makefile  print_squares.c  square.c
> cat Makefile
PROG =  print_squares

SRCS =  print_squares.c square.c

OBJS =  print_squares.o square.o

LIBS =

CC = gcc
CFLAGS = -O
LDFLAGS =

all: $(PROG)

$(PROG): $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LIBS)

clean:
        rm -f $(PROG) $(OBJS) *.mod

> make
gcc -O   -c -o print_squares.o print_squares.c
gcc -O   -c -o square.o square.c
gcc  -o print_squares print_squares.o square.o
> ./print_squares
Square of 3 is 9
> touch print_squares.c #This is the smallest possible change
> make
gcc -O   -c -o print_squares.o print_squares.c
gcc  -o print_squares print_squares.o square.o
> make clean
rm -f print_squares print_squares.o square.o *.mod
> make print_squares
gcc -O   -c -o print_squares.o print_squares.c
gcc -O   -c -o square.o square.c
gcc  -o print_squares print_squares.o square.o
>
```

Arcus-b

- Arcus-b is the cluster we are running on

- It is the main central cluster for the University, consisting of around 400 nodes, each with 16 cores and at least 64 Gbytes of memory
  - Remember 16 cores per node mean that the 64 Gbytes on a node is shared by the 16 cores
  - Important for OpenMP later on – OpenMP on arcus-b is restricted to at most 16 cores

- Important to know about for 2 reasons
  - We need to know how to manage the software environment via the module system
  - We need to know how to access those 400 nodes via the batch system

Arcus-b Software Environment

- You gain access to software via the module system
  - `module load` gives access to the software
  - `module avail` lists what is available
  - `module list` lists what is currently loaded
  - `module purge` removes everything – often useful to start from a blank sheet to avoid confusion

```
> icc print_squares.c square.c
-bash: icc: command not found
> module load intel-compilers
> icc print_squares.c square.c
> ./a.out
Square of 3 is 9
> gcc --version
gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-11)
Copyright (C) 2010 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

> module load gcc
> gcc --version
gcc (GCC) 4.8.2
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

> gcc print_squares.c square.c
> ./a.out
Square of 3 is 9
>
```

Batch Systems

- Like most clusters on arcus-b you don’t log onto the nodes that do the computation
- Instead you log onto the *login* nodes and there do you editing, compiling, file managing etc.
- And when you are ready to do the computation you send the jobs to the *compute nodes* via the batch system
- What this means is you must specify what resources you require
  - Typically number of nodes (for us always 1) and time, possibly memory
- And the batch system will send you job to an appropriate (set of) nodes once the resources become available
  - Remember the system is shared
  - You may have to wait
  - And remember these nodes are NOT connected (by default) to you screen and keyboard
- Lots of batch systems, Arcus-b uses something called SLURM

A SLURM Batch Script

Resources we need to reserve

- 1 node for 10 minutes

```
> cat script.sl
#!/bin/bash
# set the number of nodes and processes per node
#SBATCH --nodes=1
# set max wallclock time
#SBATCH --time=00:10:00
# set name of job
#SBATCH --job-name squares

# Set up the software environment
module purge
module load intel-compilers
# Run the program
./print_squares
```

Commands to run on the resources once they become available

UNIVERSITY OF OXFORD

Oxford e-Research Centre

Using the Batch System

```
> ls
print_squares  print_squares.c  script.sl  square.c
> squeue -u oerc0085
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
> sbatch script.sl
Submitted batch job 1702381
> squeue -u oerc0085
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           1702381   compute  squares oerc0085 PD       0:00      1 (None)
> squeue -u oerc0085
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           1702381   compute  squares oerc0085 CF       0:01      1 cnode1105
> squeue -u oerc0085
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
> ls
print_squares  print_squares.c  script.sl  slurm-1702381.out  square.c
> cat slurm-1702381.out
Square of 3 is 9
> 
```

- Don't worry! We will provide the batch scripts, you just need to know
  - How to use them with sbatch, squeue and scancel
  - The basics of modifying them (e.g. new commands, or longer time)

Much, MUCH more in the Practical

- We've gone through this all very quickly
- Don't worry if it's still a bit unclear, you only really learn by doing it
- And next is the practical where we will go through all this in a lot more detail

What have we learnt?

- Linux is the OS most supercomputers use
- You interact with it via the command line
- There are many compilers, we will use the Gnu and Intel ones
- Compilation is really two phases, compilation to an object file, and then linking the object files together to produce the executable
- The make utility can take advantage of this to speed up compilation, especially for large projects
- Arcus-b is the cluster we will use
- It has 16 cores per node
- The software environment is managed via the module system
- Access to the compute nodes is via the batch system

Further reading

In the next lecture…

<A synopsis of the next lecture>