An Introduction to HPC and Scientific Computing – Compiling and Batch Systems

– Practical Sessions –

Jacob Wilkins *jacob.wilkins@oerc.ox.ac.uk*
Ian Bush *ian.bush@oerc.ox.ac.uk*

Contents

1 Introduction

In these exercises we will look at compiling C programs, and introduce use of the batch system.

The practicals will be performed on the cluster. Log on using the username and password that have been provided, and the files are under the directory `notes`

Remember, to log in use:

`ssh -CX [username]@htc-login.arc.ox.ac.uk`

2 Hello World

About

This exercise introduces compiling with one of the simplest programs possible, the "hello world" code. The source files are found in the directory `hello`:

- `hello.c` – "hello world" program

Tasks

1. Before we start each question we need to set up the software environment. First issue a `module purge` command to ensure that you are stating from a clean sheet. Now we need to load the module that makes the compiler available. Throughout these exercises we shall use Intel compiler. To load this issue `module load intel-compilers`. If you have time at the end you might to try the practicals with the gnu compiler. If so instead use `module load gcc binutils`. You won't need to change any of your programs if you do this, but the makefiles will need changing as the gnu compiler accepts different flags, and uses the command gcc to invoke the compiler.
2. Edit the "hello world" program `hello.c` and try to understand it
3. Compile the program with `icc hello.c`
4. Use ls to see executable which the compilation has made, `a.out` in this case
5. Run the executable. does it give the output you expect?
6. Re-compile, this time with an appropriate flag so the executable is called `hello` Check you have done this successfully via ls, and running the new executable

3 Makefiles

About

In this exercise we introduce compiling a program which is held in multiple files, and also the use of a makefile. The example implements two different ways to orthonormalise a set of vectors, i.e. adjust the vectors so that they are all at right angles to each other, and compares the time for the two methods. Whilst rather abstract this is actually part of a number of algorithms commonly used in computational science.

The source files are found in the directory `gs`:

- `gs_opt.c` – the main code,
- `timer.c` – routines that implement the timing,
- `array_alloc.c` – routines that implement memory allocation

Tasks

1. It's good practice to make sure that you have a clean software environment at the start of each question. So again issue `module purge` and `module load intel-compilers`. Are you sure you understand what these commands are doing?
2. Compile the code with `icc gs_opt.c array_alloc.c timer.c -o gs`. Run the program and enter 1000 for both the order and number of the vectors and look at the output
3. This command is getting a bit long already with just 3 files! We have also provided a makefile to help you compile this program. Recompile the program by issuing the command `make` and run the program as before
4. Issue an `ls` command - can you see the object files?
5. Issue the command `touch timer.c` and then `make`. What happens? How is it different from the previous `make`?
6. Run the program again with the input as before. Note down the times taken for the two methods.
7. Open up the makefile with your editor of choice. In it you will see a line looking like

   ```
   CFLAGS = -O0
   ```

   This controls the flags used by the compiler. In particular the `-O` flag controls how hard the compiler works to make an executable that runs quickly. This is called "optimisation". A small number, such as zero, means little effort is put in. A large number (the maximum is typically 3) means a lot of work is put in. Change this line to read

   ```
   CFLAGS = -O2
   ```

recompile the program using `make clean` and then `make` and re-run it. How have the times changed?

8. try `-O1` and `-O3`, how does this affect the run time? Remember to use `make clean` between each recompile to make sure everything is rebuilt from scratch.

4 Batch Systems

About

We shouldn't really run any serious jobs on the front end nodes. As you have to share the front end nodes with other users the timings we have gathered so far are not that reliable, also they may impede other people's work and get you angry emails from the systems administrator. The correct way is to use the compute nodes via the batch system. In the `gs` directory you should have also noticed a file called `script.sl`, this is an appropriate batch script for the arcus-htc system and we will now use it to run the `gs` program on the compute nodes

Tasks

1. Examine the `script.sl` with an editor. Do you understand what it does? Think carefully about how it reserves the resources we need to run the program, and also about how the input is provided for the program - on the compute node there won't be a keyboard attached to provide that input!

2. Issue the command `sbatch script.sl`. What happens?

3. Examine the running job with `squeue -u <username>` replacing username as appropriate. Note the the number of the job is the same as that printed out by the `sbatch` command

4. When the job is finished use `ls` to find the output. You should find a new file in the directory called something similar to `slurm-1702775.out`, where the number will be the same as that printed out by sbatch. Examine the contents of this file.

5. Re-run the timing exercise from the previous question where you examined how fast the program ran as a function of optimisation level, and see if the ime in batch differes much from that measured on the front end

6. Rather than `slurm-1702775.out` how can you get the output written to a filename of your choice?
   Slurm's `sbatch` docs might be useful `https://slurm.schedmd.com/sbatch.html`