New Student Readme

October 1, 2008

Table of Contents

1 Code of Practice

1.1 Human Relationships

When you talk to me either during the meeting or in non-meeting hours, you should clearly show me your *own* understanding of the problem. I will not judge you for your mistakes but support you and correct your understanding. You should share you research understanding and problems. You should not act as you understand *everything* if you do not. Ultimately, when it comes to research you and me are co-workers. We stand at the same level and you should treat me as your collaborator.

1.2 Security

In the past there were some incidents involving access to various machines in our cluster and entering permission-unprotected directories as well as viewing files whose owner was different from the accessing person.

The cluster administrator is capable of tracking all actions of all users. Ignorance and lack of Unix system knowledge will not be considered as an excuse for violating these security rules.

Unless you execute the parallel computing programs in our computer cluster or belong to the cluster service providers you are expected to login and use only your *own* machine and access only your *own* or shared files.

Increased security constrains put more burden on the cluster administrator. Therefore, students demonstrating strong Unix skills might be asked to help the senior cluster administrators due to the current staff shortage.

Please read, sign and return to me the local security policy document which might be found on my website under `http://www.cs.man.ac.uk/~fumie/tmp/rule.pdf`

1.3 Biweekly Report

The biweekly report document should have the following form:

- Date
- Work achieved during the last two weeks *relative to the objectives set two weeks ago*. If the objectives were met, state the fact clearly. Otherwise analyse your activity and state the reasons why the objectives were not met.
- Work achieved during the last two weeks additionally to the pre-set objectives.
- Future work plan and objectives for the next two weeks.

Include results and discussions as well as the critical analysis. Reference published work if appropriate. When the biweekly report is completed, send an email to me exclusively stating the directory and file name containing the report.

Always use the *same file* for the biweekly report. Continuing the document will result in your entire work progress stored in a single location. This will be helpful in the future for the work review, progress report and thesis writing.

Do not send the PDF file to me. Stating the directory name and the file name is enough. Biweekly report hand-in deadlines are the *first* and the *third* Wednesdays of the month. There is no biweekly submission on the 5th Wednesday. There is no link between the biweekly report and the

weekly meeting. There will be no reminder email on the biweekly report. Missing the biweekly report hand-in date for more than 2 days is regarded as "No Progress". Please control your diary appropriately.

2 General Usage of Our Beowulf Cluster

2.1 Linux and Command-Line Utilities

Linux as all Unix systems is famous for the powerful command line utilities it offers. Refer to the following Wikipedia articles if you have little knowledge about Linux and Command-line interface:

```
http://en.wikipedia.org/wiki/Linux
http://en.wikipedia.org/wiki/Command_line_interface
```

Refer to the openSUSE 10.3 Reference for the introduction to the important Linux commands and secure network operations:

```
http://www.novell.com/documentation/opensuse103/pdfdoc/
opensuse103_reference/opensuse103_reference.pdf
```

Reading entire Chapters 19 and 36 is strongly advisable:

19: Working with the Shell	pp. 271–296
36: SSH–Secure Network Operations	pp. 589–596

The most important reference Sections are:

19.3: Important Linux Commands	pp. 280–290
36.2: The ssh Program	p. 590
36.3: spc–Secure Copy	p. 590

Refer to the Wikipedia article below for a list of frequently used Unix commands:

```
http://en.wikipedia.org/wiki/List_of_Unix_programs
```

Bash is a command-line interpreter provided with the majority of Linux distributions. It allows you to execute the command-line utilities and programs:

```
http://en.wikipedia.org/wiki/Bash
http://www.linuxconfig.org/Bash_scripting_Tutorial
http://bash-hackers.org/wiki/doku.php
http://wooledge.org:8000/BashPitfalls
```

If you face the `bash` message "command not found" your shell did not recognise the command you executed. It might occur because of the incorrect spelling of command name, e.g. "gfrtran" instead of "gfortran". In the other case the executed command might not be present in your system or you do not have sufficient user rights to run it.

2.2 X11 Forwarding

A short step-by-step guide explaining how to export graphical output from a remote Linux machine to your local Linux computer might be found under:

```
http://www.linux-tip.net/cms/content/view/302/26/
```

2.3 Handling TAR-files

Compressing and decompressing data to and from TAR-file archives:

```
tar -cjvf <archive_name> <file_list>
tar -cjvf archive.tar.bz2 file1.txt file2.txt
```

This command will compress two files into an archive. You may also use common Unix wildcards and type "*" or "file?.txt" to compress all files in the directory or all files with a variable single symbol value.

Follow the command below to decompress a TAR-archive file into a current directory:

```
tar -xjvf <archive_name>
tar -xjvf archive.tar.bz2
```

2.4 Firefox Problem Tied with NFS

Home directories in our cluster are mounted via NFS and backed up daily. NFS might cause problems running Firefox web browser on a certain machine. Please follow the steps below to restart Firefox:

1. `cd ~/.mozilla/firefox/<letter_sequence>.default`
2. `ls -lrt lock` to find the machine's IP address where you run Firefox at the moment
3. `rm lock` to delete the lock
4. Login to the machine where you were running Firefox
5. Find the process number which is related to Firefox
6. `kill` all processes associated with Firefox on that particular machine

3 Programming

3.1 General Programming

For the introduction to the basic programming concepts and style see:

```
http://www.cs.man.ac.uk/~fumie/tmp/building_programs.pdf
http://en.wikipedia.org/wiki/Programming_style
```

Programming skills are acquired in practise and experimentation. Do not rely on pure book reading or asking people about each error message.

Programming as well as LaTeX typesetting and shell scripting involves *debugging* of your code so that you can compile the program without errors and obtain the expected result.

By facing and tackling the error messages from compilers you actually learn the techniques for programming. When you are stuck with the compilation or receive strange error messages, which you can not fix, you can produce a little program which mimics your current problem in the large production code.

It is much simpler to debug and fix the "extracted code". Once you make it work you can transfer the problem solution to the major code.

If you still can not solve the problem, search for the exact error message on the Internet using Google. This activity will also give you an extra knowledge around your problem.

Please do not ask other students in the group to solve your programming problems unless you tried all the above methods. If you need to ask a question be as specific as possible. Respect other people's time since all of us work under tight schedules.

3.2 Fortran

GNU Fortran compiler `gfortran` is used in our cluster. Prior to running any program code, it has to be compiled. Here are some links which will help you further:

General information on compilers:
`http://en.wikipedia.org/wiki/Compiler`

Basic compilation and running of Fortran code:
`http://www.mesoscale.iastate.edu/agron505/compile.htm`

More complete guide on compilation and running programs in Fortran:
`http://www.fortran.gantep.edu.tr/compiling-g95-basic-guide.html`
`http://gcc.gnu.org/wiki/GFortranGettingStarted`

Compiler options are described under:
`http://linux.die.net/man/1/gfortran`

The wrap-up script `mpif90` binds `gfortran` and `mpich` to provide a convenient complication tool for the distributed memory architecture programs.

Use `mpif90intel` instead of `mpif90` if you need to test compile your code with the Intel Fortran compiler.

The 32-bit machines in our cluster are also suitable for running parallel programs.

Output to `stdout` and `stderr` is not printed out immediately after calling the `print` statement in GNU Fortran. Instead the output is buffered (which results in output delays) unless the program believes the output is done to a terminal. To force immediate output from your program you have to use the `call flush` statement in the code straight after the program action, e.g.:

```
c = a + b
print *, ''c: '', c
call flush

d = a * b
print *, ''d: '', d
call flush
```

3.3 Problem Solving

The best way to approach a problem is to decompose it into smaller simpler parts. Decide on what parts you can solve, which tools you are going to use for a solution. Then think and write a draft algorithm for your solution. It can be made on paper. Afterwards you can start implementing it using the tools you have chosen. It is always a good idea to implement the algorithm in smaller chunks. Where each chunk is performing a specific function of the main algorithm. You can always write a very basic program testing a specific tool or algorithm functionality. If your prototype works correctly you can increase its functionality and integrate into the main program.

To solve the file opening and reading problem in Fortan. Decompose it into simpler tasks:

- generating target file name,
- opening target file,
- reading data required,
- closing target file

All of those tasks can be written as 4 separate Fortran programs. Afterwards you can put it all together into one main code. Finally, you may address the loop issue–and wrap the 4 functions up with a triple loop for i, j and k values.

As a rule of thumb try to search for problem solutions on the Internet using Google. This is the quickest way to find answers. Extract the error message of your code, paste it in Google and search for answers. Finding information on your own will make you learn and remember. It is much better for you, than me telling you the answer straight away.

3.4 Running MPI Code

Before running a parallel program with MPI an MPI service (daemon) has to be launched. Running on the background this daemon provides Message Passing functionality to your programs.

Starting an MPI daemon:

```
mpd &
```

This daemon has to be started on each machine, where you intend to execute parallel program. All MPI daemons have to be connected to each other. This daemon ring connection is set up using the port number and IP address of the first machine, where "mpd" runs. This computer is called "MPI-host".

Find out port number and machine name of mpd-host:

```
mpdtrace -l
```

The output is the following:

```
<mpd_host_name>_<port> (<IP_address>)
130-88-154-30_20115 (130.88.154.30)
```

Extract port number, e.g. 20115 and IP address, e.g. 130.88.154.30.

Login to another machine in the cluster using SSH:

```
ssh -XC -P 22111 <username>@<IP_address>
ssh -XC -P 22111 maxim@130.88.154.30
```

Enter your password on SSH prompt. Now you have to start MPI daemon on this machine using port number and IP address extracted from the MPI-host.

```
mpd -h <IP_address>  -p <port>
mpd -h 130.88.154.30 -p 20115
```

The MPI environment is ready. Time to compile a parallel code. Compiling MPI code involves calling of a wrapper script "mpif90". This script launches ordinary Fortran compiler (gfortran), but also links the code with MPI libraries.

```
mpif90 <source> -o <executable>
mpif90 source_code.F90 -o executable_code
```

This command will compile a Fortran source code `source_code.F90` into an executable file called `executable_file`.

To run a parallel code on a number of processors write:

```
mpiexec -n <proc_number> <executable>
mpiexec -n 2 ./executable_file
```

In general you should keep your program source code in your home directory because it will be backed up every day. Designate a *different* output directory to e.g.

```
/local/<user_name>/<prog_name>/
/local/maxim/fdtd/
```

Specifying a local output directory will force your code to use the hard drive of a certain machine *only*. Using the home directory for output will synchronising the data files between all machines of the cluster and considerably slow down other people's computers. Please refrain from doing this.

The IBM Redbook on "Practical MPI Programming" is a recommended reading. It is available for download under:

```
http://www.redbooks.ibm.com/redbooks/pdfs/sg245380.pdf
```

The most important Chapters are:

1: Introduction to Parallel Programming	pp. 1–10
2: Basic Concepts of MPI	pp. 11–40
Appendix B: Frequently Used MPI Subroutines Illustrated	pp. 161–206

Although the environment setting and program compilation are strongly site-dependent, you can get a flavour of parallel programming by reading the following information:

```
http://www.osc.edu/supercomputing/training/
http://www.hpcwire.com/
http://www.insidehpc.com/
http://www.clustermonkey.net/
```

4 LaTeX

An article on Wikipedia gives a short introduction on LaTeX – a document markup language extensively used in a scientific domain:

```
http://en.wikipedia.org/wiki/LaTeX
```

Wikipedia also holds an entire LaTeX textbook which you can browse by topics:

```
http://en.wikibooks.org/wiki/LaTeX
```

There's a University LaTeX users list. If you're interested, send an email from your University email account to `LISTSERV@listserv.manchester.ac.uk` with the body text "subscribe latex-users".

5 To BSc and MSc Students

Practical skills such as the operating system, programming and documentation language knowledge are not going to be forced by me unless your work has a significant impact on the mainstream of our research.

In this case, you can learn Unix, Fortran, LaTeX and shell-scripting including the command-line utilities as `awk`, `sed`, `tgif` and `gnuplot` *independently*.