The AC3 High Performance Computing Facility:
An introductory guide

A/Prof. Tim Marchant,
School of Mathematics and
Applied Statistics and IMMaCS
Introduction

- the Australian Centre for Advanced Computing and Communications (ac3), is based at the Australian Technology Park, Redfern.

- it is a partnership between the State Government, ATP, various universities (including the University of Wollongong) and industry partners

- the ac3 website is www.ac3.com.au

- they have 3 machines from different companies, which hence have different architectures and operating systems

- Clare, 64 processor SGI ORIGIN 2400, this is a shared memory machine, with one central memory and many processors. Easy to use and suited to using OpenMP

- Mudgee, 68 processor IBM SP2, this is a distributed memory machine. Suited to using MPI or PVM. The IBM operating system is hard to use.
- **Hunter**, 2 processor NEC SX5, this is a vector processor. Runs at 8GFlops, some 10 times faster than a single processor on Clare
- the differing architectures of each machine makes them suited to different parallelisation techniques
- the ac3 website has a table to help you decide what machine best suits your computer code
- however, unless you have specialist needs or skills, choose Clare, as it is the simplest machine to use
- the rest of this talk will be based on using Clare
- the Silicon Graphics web-site is www.sgi.com
- a single processor on Clare is about 5 times faster than worner
Software

- freeware and vendor supplied software is available
- Fortran 77 and 90, C, C++, Java2
- a vendor supplied scientific library, SCSL; this includes linear equation solvers and signal processing routines, such as FFT and convolution
- the netlib library is another source of free software packages
- Scilab, a free linear algebra package, accepts commands using MATLAB syntax
- no commercial packages available yet. FEM and CFD software, and statistical packages are needed
My work

- I am involved with two projects which use Clare
- microwave heating: this involves solving Maxwell’s equations and the heat diffusion equation in a waveguide or cavity
- a large matrix system must be repeatedly solved
- runs on Worner take many days, on Clare with parallelisation they take a day or so
- about a 40% speed up is obtained by parallelisation of this code, compared to a single processor
- solitary wave collisions: this involves solving an extended modified Korteweg-de Vries equation using an implicit finite-difference method.
- only limited speed-up obtained by parallelisation but still runs a lot faster than on Worner, even in single processor mode. Each job takes up to an hour
Getting Started

- to obtain an account, download an application form from the website
- send the completed form to Tim Marchant
- unless you have existing HPC expertise, and a specific application which requires one of the other machines, choose Clare as your machine
- Clare is compatible with Prof. Tsoi’s Silicon Graphics machine, which is housed locally
- I will email you when your account is set up and your token is ready for collection
Logging on to Clare

• **Method 1**
  - this method is quicker, but the DISPLAY environment is lost. This means you cannot view pictures or use graphical interfaces
  - type `$ telnet clare.ac3.com.au` at the prompt type your username and then enter the 6-digit code into your token. The token’s response is then used as the password for the firewall
  - once through the firewall you can login to Clare

• **Method 2**
  - more complicated but allows the DISPLAY environment to be preserved.
  - first you must login to the firewall. Use the command
    
    `$ /usr/bin/telnet granite-belt.ac3.com.au 1234`
• the usual telnet restricts you to the default port. To use port 1234 you need to use /usr/bin/telnet. You must ask ITS for permission to use this command

• the token must be used to login to the firewall, as for Method 1. Then type `ssh at the prompt

• open another window on your workstation. Type
  
  `$ ssh -p 222 clare.ac3.com.au -o StrictHostKeyChecking=no`

• after the first time you login the HostKeyChecking flag is not needed

• now your first window will prompt you for a reply (of yes)

• the firewall then allows you to login to Clare on your second window. Both windows must then be kept open
The batch queue and ftp

- to transfer data to/from your Wollongong account you must login to Clare first, then type, $ ftp ftp.uow.edu.au
- you can just run short jobs, < 30 minutes of CPU, in the background
- longer jobs need to be run on the batch queue. Create an executable script file called batchrun. This can be one line at its simplest, e.g. /home/tim/prog1
- $ qsub batchrun will place the job on the queue
- other useful commands are qstat -a, to view the queue, qdel, to delete the job, etc
**Parallelisation techniques**

- there are many different parallelisation methods for your FORTRAN or C++ code
- OpenMP compiler directives: these suit shared memory, multiprocessor, machines, like Clare
- these directives can be manually inserted in the code to parallelise DO loops. They distribute the calculations in the loop to a number of processors
- automatic parallelisation: with this option the compiler inserts openMP compiler directives automatically
- you can insert apo compiler suggestions and directives to improve the automatic parallelisation
- Message passing: suits distributed memory machines, such as Mudgee. PVM or MPI commands are the standard ways for passing messages between processors. Complicated to use!
Parallelisation of Fortran code

- similar for C++ code
- to run your code on one processor,
  $ f77 -o test test1.f -O3$
  $ nohup timex test1 &$
- -O3 option optimises code on a single processor
- timex reports the CPU used by your program
- I shall describe the automatic parallelisation option (apo) as it is easy to use
- this option inserts openMP compiler directives automatically
- compiler directives and assertions can be used to guide the apo, if you do not like the outcome
- the directives for the apo are not the same as the manual OpenMP compiler directives
the apo coexists with manual OpenMP commands however

so you can manually parallelise some loops while automatically parallelising others

compile using

$ f77 -o test test1.f -O3 -apo -apokeep

the -apo flag is the automatic parallelisation option

the -apokeep flag generates test1.m, which is the Fortran code with the openMP directives added by the apo

before running your job you must type

$ sentenv OMP_NUM_THREADS n, where n is the number of processors you want to use
Complier directives and assertions

- automatic parallelisation only parallelises loops when it is certain than it is safe to do so
- it will not parallelise loops if there is a dependency between the loops, function calls, GOTO statements, array subscripts which are too complicated etc.
- also the wrong loop may be parallelised for a number of reasons. It is usually more efficient to parallelise the outer loop, rather than the inner loop
- you can insert apo complier directives into your code to help the parallelisation
- there are 8 different directives and they are described in the MIPSpro Fortran 77 programmers guide
- these provide extra information for the apo and can lead to a much more efficient parallelisation
**Example 1**

- test1.f is a simple program comprising nested DO loops
- the apo option parallelises the inner loop, as the \( x(i) \) are used in the calculation of the \( y(i) \)
- CPU time (seconds): 65 (n=1), 34 (n=2), 23, (n=8)
- the outer loop is usually more efficient to parallelise, especially if its loop count is bigger
- a compiler directive
  
  C**$** ASSERT DO (CONCURRENT)
  
  instructs the compiler to ignore the data dependency, so the outer loop is parallelised instead
- CPU time (seconds): 65 (n=1), 33 (n=2), 10, (n=8)
Example 2

- test1.f contains nested loops and a function call
- the apo option parallelises the inner loop as loops containing function calls will not be parallelised
- to parallelise the outer loop also need the compiler directive, C**$** ASSERT CONCURRENT CALL
- this tells the compiler the function call is safe to parallelise
- need to be careful with COMMON blocks and global variables
- the CPU times are similar to Example 1