# Tools and more for High-Performance Computing



## Overview

- Environment & account setup
- Compilers
- IDEs
- Libraries
- Make & Makefiles
- Version control
- Data analysis tools: awk & perl
- Visualization tools

IMM

02614 – Tools and more

DTU

# The DTU computer system




IMM

02614 – Tools and more

DTU

# The DTU computer system

- Campus servers:
  - 1 SF E25K (72 US-III 1050/1200 MHz)
  - 2 SF E6900 (24 US-IV 1200 Mhz)
  - 10 SF V440 (4 US-IIIi 1062 MHz)
  - 1 SF 6800 (24 US-III 1200 MHz)
  - 15000+ users (students + employees)
- HPC servers:
  - 1 SF E25K (48 US-IV+ 1800 MHz)
  - 1 SF 6800 (24 US-IV+ 1800 Mhz)
  - 200 HPC users

IMM

02614 – Tools and more

DTU

# The DTU computer system

- most applications on the system are started by a load-balancing system
- there are different CPU types, clock frequencies, amounts of RAM, etc
- this is a multi-user system(!)
- if you want to compare performance numbers:
  - make sure to be on the same system/machine
  - check the load (`uptime` command) – and ...
  - ... check the CPUs on-line (`cpucount` command)



IMM

02614 – Tools and more

DTU

# Access to the system

- On Campus:
  - SunRay terminals in the computer rooms at DTU (databars). Get a smart-card to be more flexible.
- Remote access:
  - Secure SHell (ssh) connection.
  - ThinLinc remote desktop session:
    - download ThinLinc client from www.thinlinc.com
    - connect to thinlinc.gbar.dtu.dk
    - preferred way, if you work a lot with GUIs

IMM

02614 – Tools and more

# Account setup

- Special HPC setup on the G-bar computers
  - add the line

    /appl/htools

    to your ~/.grouprc file and log out and in again
- This initializes the environment for you, such that you get access to the compilers and tools needed

DTU



IMM

02614 – Tools and more

# Compilers

- Sun Studio compilers & tools
  - version 11 (default) – version 12 is coming soon
  - version 8, 9 & 10 still on the system
  - use `init.ssN` (N = 8,...,11) to change version
- GNU Compilers (C/C++)
  - version 3.4.3(default)
  - use `init.gcc`
- GCC for SPARC (version 4.0.2)
  - use `init.sungcc`

DTU

# IDEs

- Sun Studio (`sunstudio`)
  - Compilers (Fortran, C/C++)
  - Debugger (`dbx`), analysis tools – more later
- Codeforge (`codeforge`)
- Graphical debuggers:
  - Totalview (`totalview`)
  - Data Display Debugger (`ddd`)
    - GUI front-end to either dbx or gdb





# Libraries

- Available Scientific Libraries:
  - Sun Performance Library (optimized)
    - BLAS, CBLAS, LAPACK, FFT, ...
  - GNU Scientific Library (GSL)
    - CBLAS, LAPACK, FFT, ...
  - NAG Library (Mark 20)
    - see http://www.hpc.dtu.dk/~gnag/
  - IMSL
  - ...

IMM

02614 – Tools and more

DTU

# Make & Makefiles

A tool for building and
maintaing software projects



IMM

02614 – Tools and more

DTU

# Make – The ideas behind

- maintain, update and regenerate groups of programs
- useful tool in multi-source file software projects
- can be used for other tasks as well, e.g. typesetting projects, flat-file databases, etc
- in general:  every task that involves updating files (i.e. result) from other files (i.e. sources) is a good candidate for make

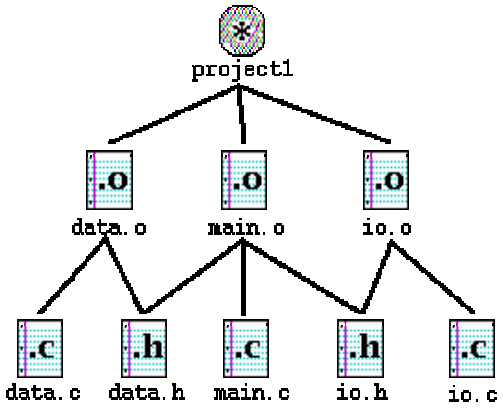
# Make – The ideas behind

Dependency graph:

result (executable)

intermediate level

source file level







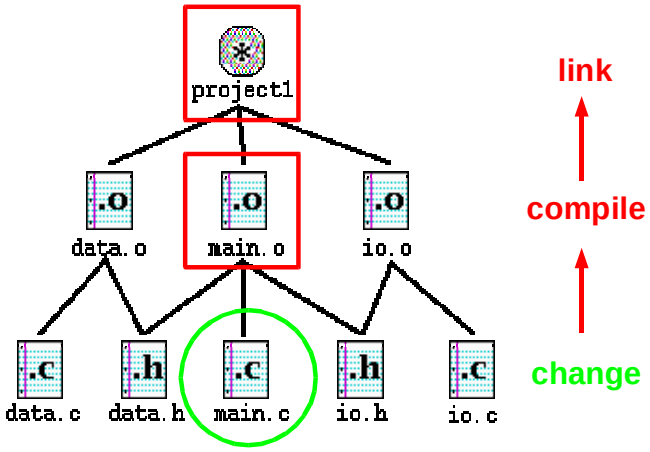

# Make – The ideas behind

Dependency graph:

result (executable)

intermediate level

source file level



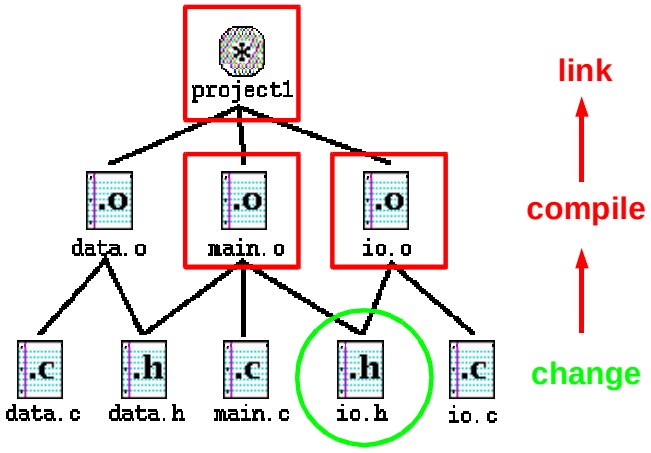
# Make – The ideas behind

Dependency graph:

result (executable)

intermediate level

source file level








# Make – The ideas behind

- Compiling by hand:
  - error prone
  - easy to forget a file
  - typos on the command line
- There is a tool that can help you:

**make**

IMM

02614 – Tools and more

DTU

# Make – The ideas behind

Things 'make' has to know:

- file status (timestamp)
- file location (source/target directories)
- file dependencies
- file generation rules (compiling/linking)
  - general rules ( .c → .o )
  - special rules ( io.c → io.o )
- tools (compilers, etc.)

- filesystem - Makefile - environment



IMM

02614 – Tools and more

DTU

# Makefile – rulesets...and more

- make needs a set of rules to do its job
- rules are defined in a text file – the *Makefile*
- standard names: Makefile or makefile
- non-standard names can be used with the '-f' option of make: `make -f mymf` ...
- preview/dryrun option: `make -n` ...

# Makefile – rulesets...and more

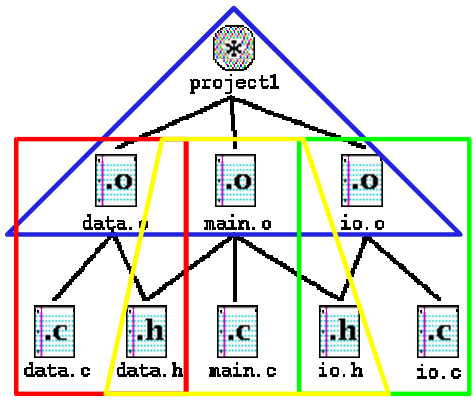
There are two major object types in a Makefile

- targets
  - definition by a ":"
  - followed by the dependencies (same line)
  - followed by lines with the commands to execute
- macros
  - definition by "="
  - single line (use "\" to extend lines)
- ... and comments: (lines) starting with #





# Makefile – rulesets...and more




```
project1: data.o main.o io.o
    cc data.o main.o io.o -o project1
data.o: data.c data.h
    cc -c data.c
main.o: data.h io.h main.c
    cc -c main.c
io.o: io.h io.c
    cc -c io.c
```

IMM

02614 – Tools and more

# Makefile – rulesets...and more

target

dependencies

```
project1: data.o main.o io.o
    cc data.o main.o io.o \
    -o project1
    echo "Done."

data.o: data.c data.h
    cc -c data.c

# the main program
main.o: data.h io.h main.c
    cc -c main.c
```

TAB !!!

command(s) to execute

comment line

DTU



IMM

02614 – Tools and more

# Makefile – rulesets...and more

```
# Sample Makefile
CC      = gcc
OPT     = -g -O3
WARN    = -Wall
CFLAGS = $(OPT) $(WARN)    # the C compiler flags
OBJECTS   = data.o main.o io.o

project1 : $(OBJECTS)
     $(CC) $(CFLAGS) -o project1 $(OBJECTS)

clean:
     @rm -f *.o core

realclean : clean
     @rm -f project1

# file dependecies
data.o : data.c data.h
main.o : data.h io.h main.c
io.o   : io.h io.c
```

Macro definitions

Macro reference

Where are my rules for compiling the .o files?

DTU

IMM

02614 – Tools and more

DTU

# Makefile – rulesets...and more

Running make:

```
bohr $ make
gcc -g -O3 -Wall   -c -o data.o data.c
gcc -g -O3 -Wall   -c -o main.o main.c
gcc -g -O3 -Wall   -c -o io.o io.c
gcc -g -O3 -Wall -o project1 data.o main.o io.o
```

How did **make** know how to build data.o, ... ?



IMM

02614 – Tools and more

DTU

# Makefile – rulesets...and more

built-in data base of "*standard rules*" and "*standard macros*":

- known rules:
  - compile .o files from a .c/.cpp/.f/... source file
  - link executables from .o files
- pre-defined macros:
  - CC, CFLAGS, FC, FFLAGS, LD, LDFLAGS
- view with `make -p -f /dev/null` (long listing!)

# Makefile – rulesets...and more

```
# GNU Make 3.80
# Variables
...
# default
OUTPUT_OPTION = -o $@
# makefile (from `Makefile', line 3)
CC = gcc
# environment
MACHTYPE = i686-suse-linux
# makefile (from `Makefile', line 6)
CFLAGS = $(OPT) $(WARN)
# makefile (from `Makefile', line 4)
OPT = -g -O3
# makefile (from `Makefile', line 5)
WARN = -Wall
# default
COMPILE.c = $(CC) $(CFLAGS) $(CPPFLAGS) -c
# makefile (from `Makefile', line 8)
OBJECTS = data.o main.o io.o
...
```



IMM
02614 – Tools and more
DTU

# Makefile – rulesets...and more

```
...

# Implicit Rules
.c.o:
#  commands to execute (built-in):
    $(COMPILE.c) $(OUTPUT_OPTION) $<

...

data.o: data.c data.c data.h
#  Implicit rule search has been done.
#  Implicit/static pattern stem: `data'
#  Last modified 2004-08-27 10:08:56.008831584
#  File has been updated.
#  Successfully updated.
#  commands to execute (built-in):
    $(COMPILE.c) $(OUTPUT_OPTION) $<
```

IMM

02614 – Tools and more

DTU

# Makefile – rulesets...and more

Practical hints:

- preview/dryrun option: `make -n ...`
- switch off built-in rules/macros:
  `make -r ...`
- check the known suffixes (.SUFFIXES) and implicit rules for your source files, e.g. does gmake still fail for .f90/.f95
- add suffixes needed: `.SUFFIXES: .f90`



IMM

02614 – Tools and more

DTU

# Makefile – rulesets...and more

Practical hints (cont'd):

- be aware of timestamps (Network-FS)
- override macros on the command line:

```
bohr $ make
gcc -g -O3 -Wall     -c -o data.o data.c
gcc -g -O3 -Wall     -c -o main.o main.c
gcc -g -O3 -Wall     -c -o io.o io.c
gcc -g -O3 -Wall-o project1 data.o main.o io.o

bohr $ make CFLAGS=-g
gcc -g    -c -o data.o data.c
gcc -g    -c -o main.o main.c
gcc -g    -c -o io.o io.c
gcc -g -o project1 data.o main.o io.o
```

IMM

02614 – Tools and more

DTU

# Makefile – rulesets...and more

Special variables/targets:

- the first target in Makefile is the one used when you call make without arguments!
- automatic variables:
  - $<  - The name of the first prerequisite.
  - $@ - The file name of the target of the rule.
- for more information:
  - man make
  - info make (usually gmake)



IMM

02614 – Tools and more

DTU

# Makefile – rulesets...and more

Makefile design – Best practice:

- start with the macros/variables
- call your first target “all:” and make it depend on all targets you want to build
- have a target “clean:” for cleaning up
- avoid explicit rules where possible, i.e. use redundancy

IMM

02614 – Tools and more

DTU

# Makefile – rulesets...and more

Makefile design – Best practice (cont'd):

- check your dependencies:
  - by hand
  - most C/C++ compilers can generate Makefile dependencies (see compiler documentation)
  - Sun Studio: `cc -xM1`
  - Gnu C: `gcc -MM`
  - external tool: `makedepend -Y`
  - Note: the options above ignore `/usr/include`



IMM

02614 – Tools and more

DTU

# Makefile – rulesets...and more

Common mistakes:

- missing TAB in “command lines”
- wrong variable references:

  $VAR instead of $(VAR)
- missing/wrong dependencies
- remember: each command is carried out in a new sub-shell

IMM

02614 – Tools and more

DTU

# Makefile – rulesets...and more

Makefiles – and Makefiles (from IDEs)

- Most IDEs create their own Makefiles
  - ... which are often not very smart
  - ... which are often not compatible



IMM

02614 – Tools and more

DTU

# Make and Makefiles: Labs

- There are five short lab exercises
- download from Campusnet
- unzip the file
- the exercises are in the directories lab_N
- read the README files for instructions

IMM

02614 – Tools and more

DTU

# Make and Makefiles: Labs

- makedepend:
  - if `man makedepend` does not work, use

```
man -M/usr/openwin/man makedepend
```

- Hints:
  - M_PI is a definition from <math.h>
  - sin() is a function from libm.so, so you have to link with that library (use -lm the right place)



IMM

02614 – Tools and more

DTU

# Version control

- Larger – but also simple – software projects need to keep track of different versions
- This is very useful during development, e.g. to be able to go back to the last working version
- Versioning Tools:
  - RCS – single user, standalone
  - CVS – multi-user, network based
  - Subversion – multi-user, network based

IMM

02614 – Tools and more

# Version control

- DTU has a central CVS server
  - nice tool to share and control source files
  - request access on http://cvs.gbar.dtu.dk/
  - basic introduction: http://www.gbar.dtu.dk/index.php/CVS
  - simple CVS exercise (in Danish):
    - http://www.gbar.dtu.dk/opgaver/cvs.pdf
- there will be a Subversion server in the "near" future as well

DTU



IMM

02614 – Tools and more

# Data analysis tools

- Scientific software usually produces lots of data/datafiles
- There are good tools to do (a quick) analysis:
  - awk – standard UNIX/Linux tool
  - perl – available on many platforms
- Both tools can be used
  - from the command line
  - with scripts

DTU

# Data analysis tools – awk

- awk operators:

```
Field reference:                $
 $0: the whole line - $n: the n-th field
Increment or decrement:     ++ --
Exponentiate:                ^
Multiply, divide, modulus:  * / %
Add, subtract:               + -
Concatenation:              (blank space)
Relational:                 < <= > >= != ==
Match regular expression:   ~ !~
Logical:                    && ||
C-style assignment:     = += -= *= /= %= ^=
```



IMM
02614 – Tools and more
DTU

# Data analysis tools – awk

## Examples:

- Print first two fields in opposite order:

```
awk '{ print $2, $1 }' file
```

- Print column 3 if column 1 > column 2:

```
awk '$1 > $2 {print $3}' file
```

- Print line (default action) if col. 3 > col. 2:

```
awk '$3 > $2' file
```

# Data analysis tools – awk

## Examples (cont'd):

- Add up first column, print sum and average:

```
awk '{s += $1}; END { print "sum
is", s," avg is", s/NR}' file
```

- Special keywords/variables:

```
BEGIN   do before the first record
END     do after the last record
NR      number of records
NF      number of fields
$NF     the value of the last field
```





# Data analysis tools

- Other useful standard Unix tools for data analysis:
  - sort
  - uniq
  - head, tail
  - wc
  - sed
  - ...

# Data analysis tools – perl

- Perl is a very powerful tool, that combines the features of awk, grep, sed, sort, and other Unix-tools into one language
- Good tool for more complex data analysis tasks
- Web-site: http://perl.org/
- Archive of perl programs:
  - Comprehensive Perl Archive Network – CPAN
  - http://www.cpan.org/





# Data analysis tools – perl

## Perl example script:

```
#!/usr/bin/perl

while (<>) {

    next if /^#/;       # skip comment lines
    @fields = split();  # split the line

    if ($#fields == 2 ) { # 3(!) elements
        print "$fields[0] $fields[2]\n";
    }
    else {
        print;
    }
}
```

IMM

02614 – Tools and more

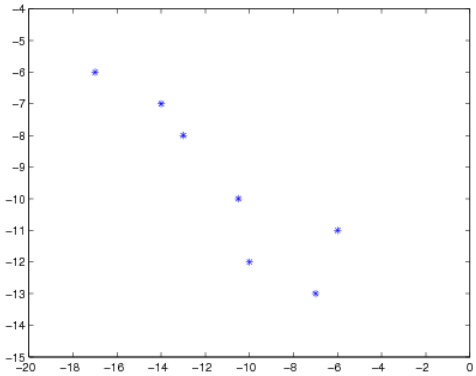
DTU

# Visualization

Visualization is an important part of Scientific Computing

Motivation: What's that?

```
A (  -17,   -6)
B (  -14,   -7)
C (  -13,   -8)
D (-10.5, -10)
E (   -6, -11)
F (   -7, -13)
G (  -10, -12)
```



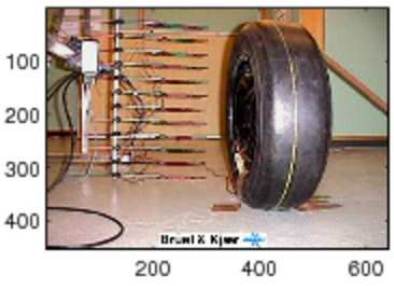


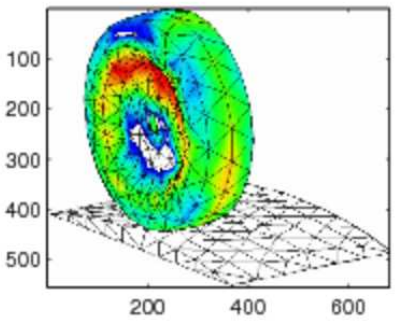

IMM

02614 – Tools and more

DTU

# Visualization



# Visualization

- Simple tools to visualize data:
  - Gnuplot (`gnuplot`)
    - command based, flexible
    - good for scripting, batch analysis
    - limited graphics (not always suitable for publishing)
  - Grace (`xmgrace`)
    - GUI-based
    - difficult to do scripting, batch analysis
    - very good graphics (publication-ready)
  - ... or whatever tool you like/prefer

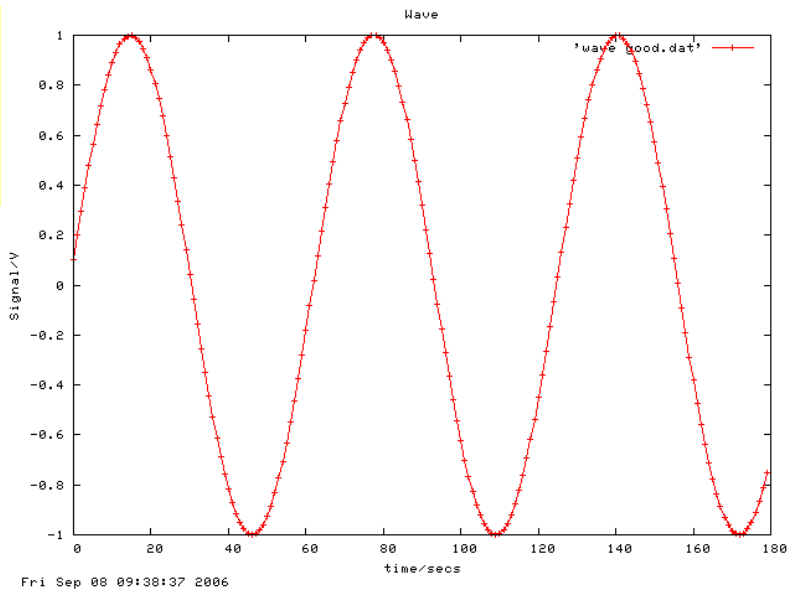




# Visualization

## Gnuplot example:

```
gnuplot>
gnuplot>
gnuplot>
gnuplot>
gnuplot>
```



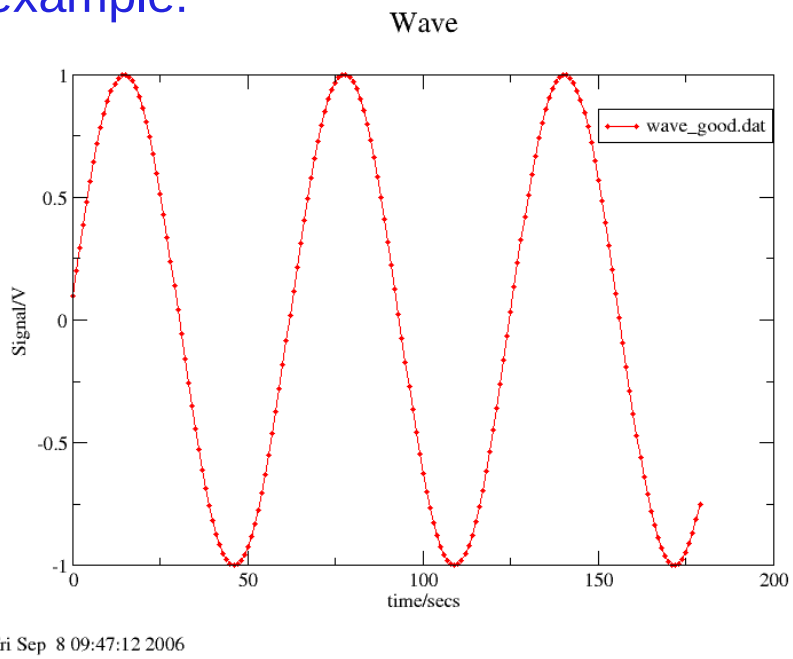
IMM

02614 – Tools and more

DTU

# Visualization

## Grace example:






IMM

02614 – Tools and more

DTU

# Visualization

- Best practice:
  - label the axes
  - use legends (and titles)
  - use the right scaling
    - a plot of a circle should be a circle
  - don't overload figures with information – use more figures instead
  - colors are useful – but can also be confusing

# Data analysis – lab exercise

- download the file wave.zip from Campusnet
- follow the instructions in wave.readme
- Goal:
  - get used to awk (choose perl, if you like or know it already)
  - get used to either Gnuplot or Grace (or the tool you know/like)