

Tools and more for High-Performance Computing

Overview

- ❑ Environment & account setup
- ❑ Compilers
- ❑ IDEs, Libraries
- ❑ Make & Makefiles
- ❑ Version control
- ❑ Data analysis tools: awk & perl (self study)
- ❑ Visualization tools (self study)
- ❑ Resource Managers

The DTU computer system



January 2021

02614 - High-Performance Computing

3

The central DTU UNIX system

- ❑ Application servers:
 - ❑ 8 Huawei XH620 V3 (2x Intel Xeon E5-2660v3 2.6 GHz)
 - ❑ 7 Dell PowerEdge FC430 (2x Intel Xeon E5-2670v3 2.3 GHz)
 - ❑ Scientific Linux 7.x
- ❑ Virtual GL servers:
 - ❑ 2 Lenovo ThinkSystem SR650 (2x Intel Xeon Gold 6226R 2.9 GHz + 2x NVIDIA Quadro RTX 5000)
- ❑ Desktop servers (ThinLinc):
 - ❑ 4 servers (4x AMD Opteron 6376, 2.4 GHz)
- ❑ 10000+ users (students + employees)

January 2021

02614 - High-Performance Computing

4

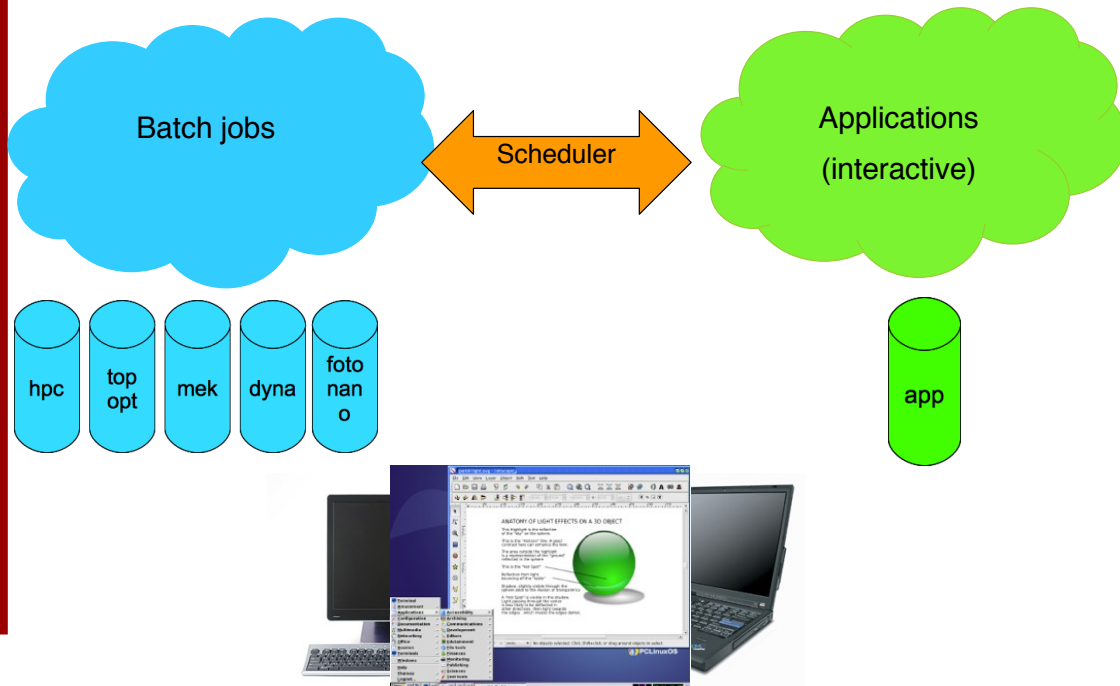
The DTU Unix systems

- ❑ HPC servers (for 'everybody'), e.g.
 - ❑ 40 IBM NeXtScale nx360 M4 (2x Xeon E5-2680v2 2.8 GHz, 128 GB memory)
 - ❑ 48 Huawei XH620 V3 (2x Xeon E5-2660v3 2.6 GHz, 128 GB memory)
 - ❑ 40 Huawei XH620 V3 (2x Xeon E5-2650v4 2.2 GHz, 256 GB memory)
 - ❑ 24 Lenovo ThinkSystem SD530 (2x Xeon Gold 6126 2.6 GHz, 192-384 GB memory)
 - ❑ 4 Lenovo ThinkSystem SD530 (2x Xeon Gold 6142 2.6 GHz, 384 GB memory)
 - ❑ 20 Lenovo ThinkSystem SD530 (2x Xeon Gold 6226R 2.9 GHz, 384-768 GB memory)
- ❑ + “private” clusters (DTU departments)

Access to the system

- ❑ Remote access, only:
 - ❑ ThinLinc remote desktop session:
 - ❑ download ThinLinc client from www.thinlinc.com
 - ❑ connect to thinlinc.gbar.dtu.dk
 - ❑ browser based: <https://thinlinc.gbar.dtu.dk/>
 - ❑ preferred way, if you work a lot with GUIs
 - ❑ Secure SHell (ssh) connection
 - ❑ login.hpc.dtu.dk or login2.hpc.dtu.dk
 - ❑ login.gbar.dtu.dk or login2.gbar.dtu.dk
- ❑ Note: those machines are login nodes!
 - ❑ no computations here, please!
 - ❑ open a ‘xterm’ (ThinLinc) or do a ‘linuxsh’ (SSH)

Our Setup



January 2021

02614 - High-Performance Computing

7

The DTU computer system

Be aware of, that ...

- ❑ this is a multi-user system(!)
- ❑ (almost) all applications on the system are started by a load-balancing queueing system
- ❑ there are different
 - ❑ CPU types,
 - ❑ clock frequencies,
 - ❑ amounts of RAM,
 - ❑ etc
- ❑ Thus, the performance will vary!!!

January 2021

02614 - High-Performance Computing

8

The DTU computer system

Comparing performance numbers:

- ❑ make sure to be on the same machine type
 - ❑ `lscpu` command
 - ❑ `echo $CPULTYPEV`
- ❑ check the load (interactive sessions)
 - ❑ `uptime` command
- ❑ check the # of CPU cores
 - ❑ `cpucount` command

Compilers

- ❑ GNU Compilers (C/C++)
 - ❑ gcc 4.8.5 (OS standard) – do not use!
 - ❑ gcc 6.3.0 ('module load gcc') – use at least this one!
 - ❑ newer versions: check with 'module avail gcc'
- ❑ Oracle Studio compilers & tools
 - ❑ version 12 upd 6 ('module load studio')
 - ❑ performance tools: collect, analyzer
 - ❑ compiler commands: `suncc`, `sunCC`, `sunf95`
- ❑ Note: 'cc' depends on the module loaded!!!
 - ❑ always use the specific names, i.e. gcc, suncc, ...

More compilers

- ❑ Intel compilers
 - ❑ version 13.0.1 ('module load intel')
 - ❑ commands: `icc`, `ifort`
 - ❑ + some extra tools
 - ❑ newer versions: check with 'module avail intel'
 - ❑ `intel/2020.4.304`
 - ❑ `intel/2019.5.281`
 - ❑ `intel/2018.3.051`
 - ❑ ...
 - ❑ use one of the newer versions above

Using modules

- ❑ modules help to organize certain Unix environment settings, e.g. `PATH`, `MANPATH`, `LD_LIBRARY_PATH`, etc. for different versions of the same application
- ❑ list available modules: `module avail gcc`
- ❑ load a module: `module load gcc`
 - ❑ loads the default version (6.3.0)
- ❑ swap a version: `module swap gcc/8.2.0`
- ❑ swap to default: `module swap gcc`
- ❑ info: <http://gbar.dtu.dk/index.php/faq/83-modules>

IDEs

- ❑ Eclipse (`eclipse4`)
- ❑ VScode (`code`)
- ❑ Oracle Studio (`sunstudio`)
 - ❑ Compilers (Fortran, C/C++)
 - ❑ Debugger (`dbx`), analysis tools – more later
- ❑ Graphical debuggers:
 - ❑ Totalview (`totalview`)
 - ❑ Data Display Debugger (`ddd`)
 - ❑ GUI front-end to either `dbx` or `gdb`

Libraries

- ❑ Available Scientific Libraries:
 - ❑ ATLAS
 - ❑ BLAS, CBLAS, LAPACK, ...
 - ❑ optimized for generic x64 platform
 - ❑ OpenBLAS
 - ❑ module load `openblas/<version>`
 - ❑ optimized for different CPU types
 - ❑ Solaris Studio Performance Library (optimized)
 - ❑ BLAS, CBLAS, LAPACK, FFT, ...
 - ❑ part of Oracle Studio
 - ❑ optimized for different CPU types

Make & Makefiles

A tool for building and
maintaining software projects

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

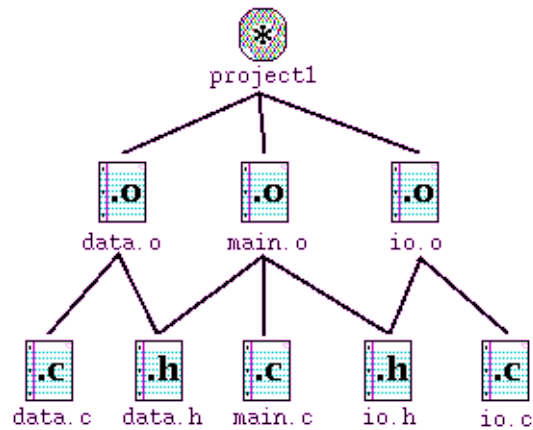
Make – The ideas behind

Dependency graph:

result (executable)

intermediate level

source file level



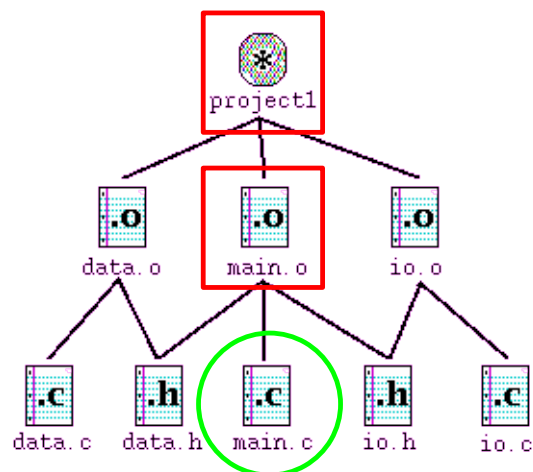
Make – The ideas behind

Dependency graph:

result (executable)

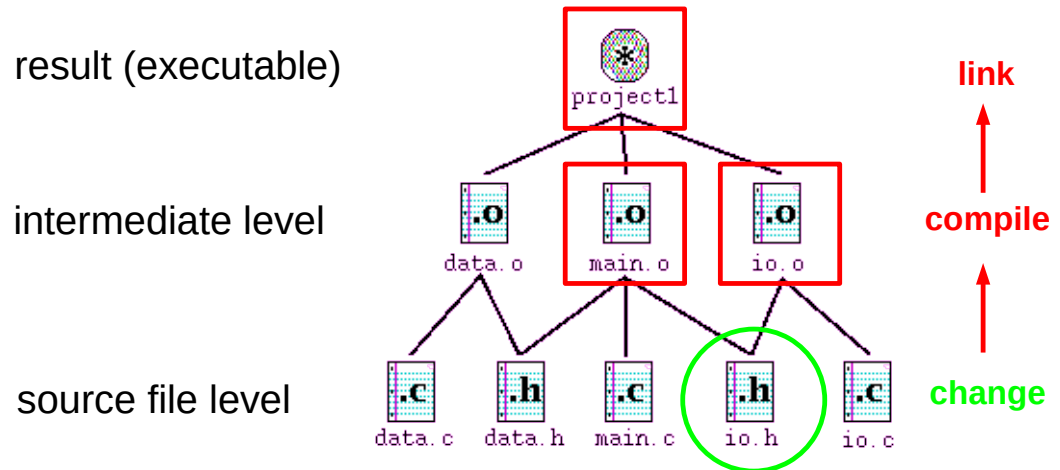
intermediate level

source file level



Make – The ideas behind

Dependency graph:



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

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

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

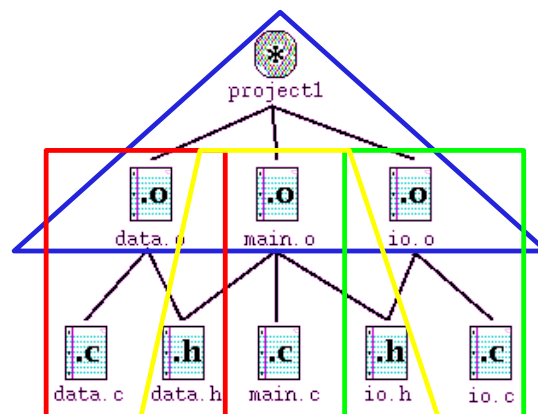
❑ targets

- ❑ 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
```

Makefile – rulesets...and more

```

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
  
```

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 dependencies
data.o : data.c data.h
main.o : data.h io.h main.c
io.o   : io.h io.c
  
```

Where are my rules for compiling the .o files?

Makefile – rulesets...and more

Running make:

```
myhost $ 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, ... ?

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
...
```

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) $<
```

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
```

Makefile – rulesets...and more

Practical hints (cont'd):

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

```
myhost $ 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
```

```
myhost $ 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
```


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

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

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: `suncc -xM1`
 - ❑ Gnu C: `gcc -MM`
 - ❑ external tool: `makedepend -Y`
 - ❑ Note: the options above ignore `/usr/include`

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

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
- ❑ make and (g)make:
 - ❑ Linux: make == gmake (GNU make)
 - ❑ Unix: make != gmake
 - ❑ if make fails, try gmake

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

Make and Makefiles: Labs

❑ 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)

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
 - ❑ git – multi-user, network based

Version control

- ❑ DTU has a central CVS server
 - ❑ nice tool to share and control source files
 - ❑ request access on <https://repos.gbar.dtu.dk/>
 - ❑ basic introduction: <http://gbar.dtu.dk/faq/34-cvs>
- ❑ ... and a Subversion (SVN) server as well
 - ❑ request access on <https://repos.gbar.dtu.dk/>
 - ❑ basic introduction: <http://gbar.dtu.dk/faq/39-svn>
- ❑ ... and some info about Git and GitLab:
 - ❑ <http://gbar.dtu.dk/faq/41-git>
 - ❑ <http://www.gbar.dtu.dk/faq/94-gitlab>

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 – (almost) standard on many platforms
- ❑ Both tools can be used
 - ❑ from the command line
 - ❑ with scripts

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:            = += -= *= /= %= ^=
```

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

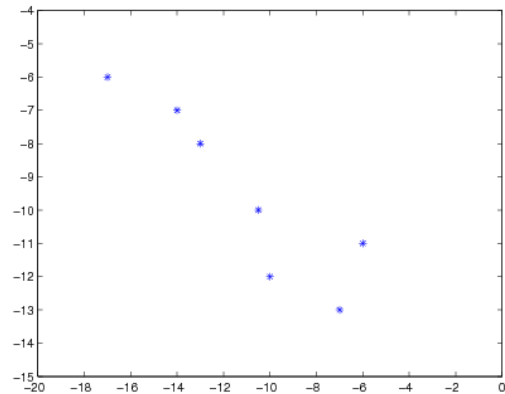
    if ($#fields == 2 ) { # 3(!) elements
        print "$fields[0] $fields[2]\n";
    }
    else {
        print;
    }
}
```


Visualization

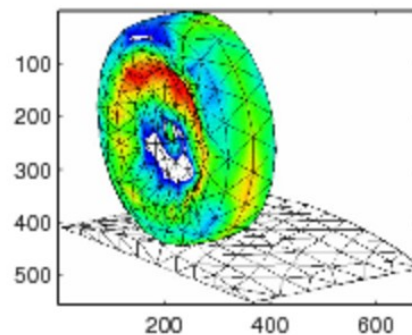
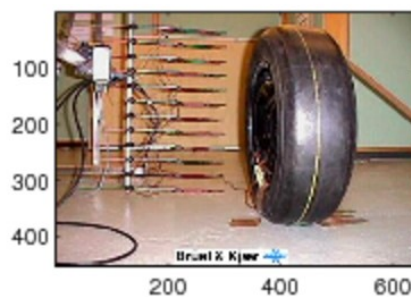
Visualization is an important part of Scientific Computing

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)
```



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

January 2021

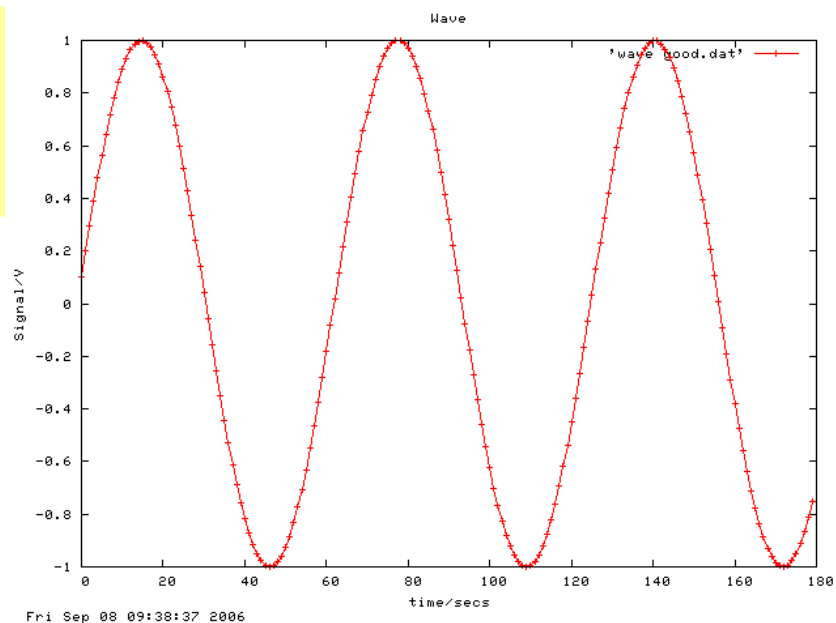
02614 - High-Performance Computing

51

Visualization

Gnuplot example:

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



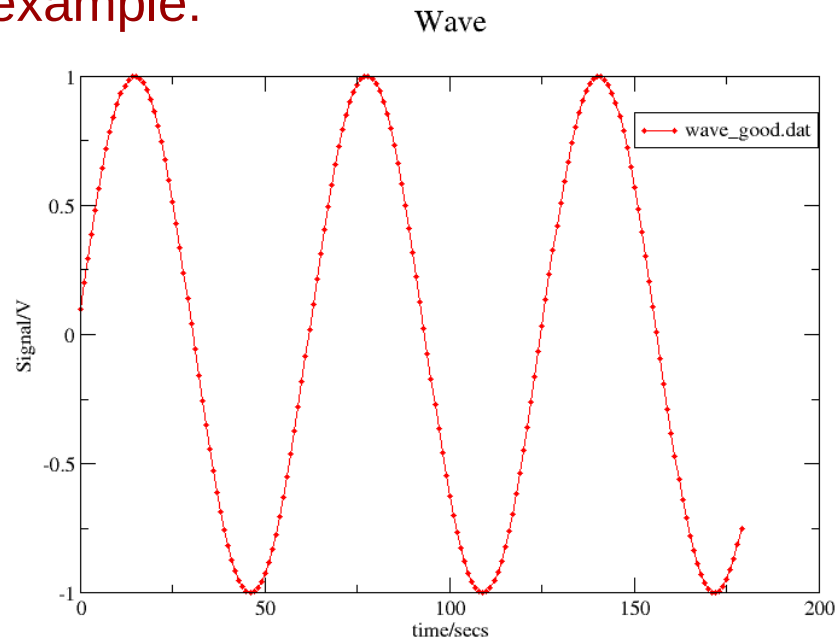
January 2021

02614 - High-Performance Computing

52

Visualization

Grace example:



Fri Sep 8 09:47:12 2006

January 2021

02614 - High-Performance Computing

53

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

January 2021

02614 - High-Performance Computing

54

And not to forget ...

... a very powerful tool/language for Scientific Computing:

- ❑ Python
 - ❑ built-in vector and matrix types (NumPy, SciPy)
 - ❑ data plot functionality (matplotlib)
 - ❑ interfaces to different languages
 - ❑ GPU support (PyOpenCL, PyCUDA)
 - ❑ and, and, and

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)

Resource Managers

To handle the workload on an HPC installation, one needs a tool to manage and assign the resources: a Resource Manager – sometimes also called 'batch queue system'

- ❑ Most common systems:
 - ❑ Torque/PBS (ext. scheduler, like Maui or MOAB)
 - ❑ LSF
 - ❑ Grid Engine
 - ❑ Slurm

Resource Managers

Before submitting a job, one has to specify the resources needed, e.g.

- ❑ # of CPUs/cores
- ❑ amount of memory
- ❑ expected run time (wall-clock time)
- ❑ other resources, like disk space, GPUs, etc

This is done in a special job script and is system (RM) dependent – but very similar for all RMs.

Resource Managers

The simplest job script:

```
#!/bin/bash
sleep 60
```

submit.sh

```
$ bsub < submit.sh
```

```
Job <702572> is submitted to default queue <hpc>.
```

```
$ bstat
```

JOBID	USER	QUEUE	JOB_NAME	SLOTS	STAT	START_TIME	ELAPSED
702572	gbarbd	hpc	NONAME	1	RUN	Dec 13 12:17	0:00:00

```
$ bjobs
```

JOBID	USER	QUEUE	JOB_NAME	SLOTS	STAT	START_TIME	TIME_LEFT
702572	gbarbd	hpc	NONAME	1	RUN	Dec 13 12:17	00:15:00 L

```
$ ls -g
```

```
total 4
```

```
-rw-r--r-- 1 gbar 1493 Dec 13 12:18 NONAME_702572.out
```

```
-rw-r--r-- 1 gbar 22 Dec 13 12:05 simple.sh
```

Resource Managers

The simplest job script – the full story:

```
#!/bin/bash
sleep 60
```

simple.sh

```
$ bsub < simple.sh
```

```
bsub info: Job has no name! Setting it to NONAME!
```

```
bsub info: Job has no wall-clock time! Setting it to 15 minutes!
```

```
bsub info: Job has no output file! Setting it to NONAME_%J.out!
```

```
bsub info: Job has no memory requirements! Setting it to 1024 MB!
```

```
bsub info: You need to specify at least -R "usage[mem=...]"!
```

```
Job <702608> is submitted to default queue <hpc>.
```

Resource Managers

A simple job script:

```
#!/bin/bash
#BSUB -J sleeper
#BSUB -o sleeper_%J.out
#BSUB -q hpcintro
#BSUB -W 2
#BSUB -R "rusage[mem=512MB]"

sleep 60

$ bsub < submit.sh
Job <702645> is submitted to queue <hpcintro>.

$ ls -g
total 3
-rw-r--r-- 1 gbar 121 Dec 13 12:32 submit.sh
-rw-r--r-- 1 gbar 1592 Dec 13 12:36 sleeper_702646.out
```

Resource Managers

□ The output file:

```
Sender: LSF System <lsfadmin@n-62-21-20>
Subject: Job 702646: <sleeper> in cluster <dcc> Done

Job <sleeper> was submitted from host <hpclogin3> by user <gbarbd> in
cluster <dcc> at Wed Dec 13 12:34:59 2017.
Job was executed on host(s) <n-62-21-20>, in queue <hpc>, as user
<gbarbd> in cluster <dcc> at Wed Dec 13 12:34:59 2017.
</zhome/../../../../> was used as the home directory.
</zhome/../../../../02614/Batch/LSF> was used as the working directory.
Started at Wed Dec 13 12:34:59 2017.
Terminated at Wed Dec 13 12:36:00 2017.
Results reported at Wed Dec 13 12:36:00 2017.

Your job looked like:
```

```
-----
# LSBATCH: User input
#!/bin/bash
#BSUB -J sleeper
#BSUB -o sleeper_%J.out
```

Resource Managers

❑ The output file (cont'd):

❑ job summary

Successfully completed.

Resource usage summary:

CPU time :	0.28 sec.
Max Memory :	4 MB
Average Memory :	4.00 MB
Total Requested Memory :	512.00 MB
Delta Memory :	508.00 MB
Max Swap :	-
Max Processes :	4
Max Threads :	5
Run time :	65 sec.
Turnaround time :	61 sec.

The output (if any) is above this job summary.

Resource Managers

Separating output and errors:

```
#!/bin/bash
#BSUB -J sleeper
#BSUB -o sleeper_%J.out
#BSUB -e sleeper_%J.err
#BSUB -q hpcintro
#BSUB -W 2 -R "rusage[mem=512MB]"
```

```
rm nonexistent.txt
echo "Just a minute ..."
```

```
sleep 60
```

```
$ bsub < submit2.sh
```

```
...
```

```
$ ls -g
```

```
total 3
```

```
-rw-r--r-- 1 gbar 184 Dec 13 13:56 submit2.sh
```

```
-rw-r--r-- 1 gbar 63 Dec 13 13:59 sleeper_702793.err
```

```
-rw-r--r-- 1 gbar 1744 Dec 13 14:00 sleeper_702793.out
```


Resource Managers

Separating output, errors – and mail summary:

```
#!/bin/bash
#BSUB -J sleeper
#BSUB -o sleeper_%J.out
#BSUB -e sleeper_%J.err
#BSUB -q hpcintro
#BSUB -W 2 -R "rusage[mem=512MB]"
```

```
rm nonexistent.txt
echo "Just a minute ..."
```

```
sleep 60
```

```
$ bsub -N < submit2.sh
```

send summary
at end of job

```
...
```

```
$ ls -g
```

```
total 3
```

```
-rw-r--r-- 1 gbar 184 Dec 13 13:56 submit2.sh
```

```
-rw-r--r-- 1 gbar 63 Dec 13 14:04 sleeper_702814.err
```

```
-rw-r--r-- 1 gbar 18 Dec 13 14:04 sleeper_702814.out
```

Resource Managers

A simple parallel job script:

- for OpenMP (single node), using 4 cores

```
#!/bin/bash
#BSUB -J openmp_para
#BSUB -o openmp_para_%J.out
#BSUB -q hpcintro
#BSUB -W 2 -R "rusage[mem=512MB]"
#BSUB -n 4 -R "span[hosts=1]"
```

```
export OMP_NUM_THREADS=$LSB_DJOB_NUMPROC
```

```
...
```

Resource Managers

Another parallel job script:

- ❑ for MPI: two nodes, using 4 cores/node

```
#!/bin/bash
#BSUB -J mpi_para
#BSUB -o mpi_para_%J.out
#BSUB -q hpcintro
#BSUB -W 2 -R "rusage[mem=512MB] "
#BSUB -n 8 -R "span[ptile=4] "

module load mpi
mpirun ...
```

Resource Managers

more options and examples:

- ❑ see <http://www.hpc.dtu.dk/> under
 - ❑ LSF User Guides
 - ❑ http://www.hpc.dtu.dk/?page_id=2534
- ❑ do the lab exercises
- ❑ use 'man bsub', 'man bjobs', etc

Resource Managers

DTU Computing Center specific commands:

- ❑ bstat – shows the status of your jobs; use 'bstat -h' for help for other options
- ❑ classstat – shows the status of the queues, e.g. free and used cores, pending jobs, etc
- ❑ nodestat – shows the current status of all nodes (use 'nodestat hpc' for the nodes of the 'hpc' queue)
- ❑ all commands above have a help (-h) option, but no man-page!

Resource Managers

- ❑ There are a few hands-on exercises on CampusNet to get you acquainted with the batch system
- ❑ more information can be found on www.hpc.dtu.dk under **LSF User Guides**
- ❑ we have a special queue for this course, 'hpcintro', so please use '-q hpcintro' instead of '-q hpc' in your job scripts!