Introduction to PDC

2022-10-27

PDC staff

Contents

Background and infrastructure

PDC Center for High Performance Computing

- High performance computing (HPC) systems and storage facilities
- Application experts and services for academic research and business R&D
- International and national research projects and infrastructures
- HPC training workshops and courses
- One of the largest and fastest SNIC HPC systems in Sweden

History of PDC

Year	rank	procs.	peak TFlops	vendor	name	remark
2017	69	67456	2438.1	Cray	Beskow	XC40 16-core 2.3GHz
2014	32	53632	1973.7	Cray	Beskow	
2011	31	36384	305.63	Cray	Lindgren	XE6 12-core 2.1GHz
2010	76	11016	92.534	Cray	Lindgren	
2010	89	9800	86.024	Dell	Ekman	PowerEdge SC1435 Dual core Opteron 2.2GHz

History of PDC

Year	rank	procs.	peak TFlops	vendor	name	remark
2005	65	886	5.6704	Dell	Lenngren	PowerEdge 1850 3.2GHz
2003	196	180	0.6480	HP	Lucidor	Cluster Platform 6000 rx2600 Itanium2 900MHz
1998	60	146	0.0934	IBM	Strindberg	SP P2SC 160MHz
1996	64	96	0.0172	IBM	Strindberg	
1994	341	256	0.0025	Thinking Machines	Bellman	CM-200/8k

SNIC

Swedish National Infrastructure for Computing

Vetenskapsrådet

PDC is a SNIC center

- SNIC (Swedish National Infrastructure for Computing)
- SNIC is a national research infrastructure that provides a balanced and cost-efficient set of resources and user support for large scale computation and data storage to meet the needs of researchers from all scientific disciplines and from all over Sweden (universities, university colleges, research institutes, etc).
- SNIC is funded by the Swedish Research Council (VR-RFI) and the 10 participating universities: Chalmers, GU, KI, KTH, LiU, LU, SLU, SU, UmU, and UU.

Collaboration with Industry

PDC's largest industrial partner is Scania. The figure shows a volume rendering of the instantaneous velocity magnitude on the leeward side of a Scania R20 Highline truck at crosswind conditions. (Source: Scania)

Collaboration with industry

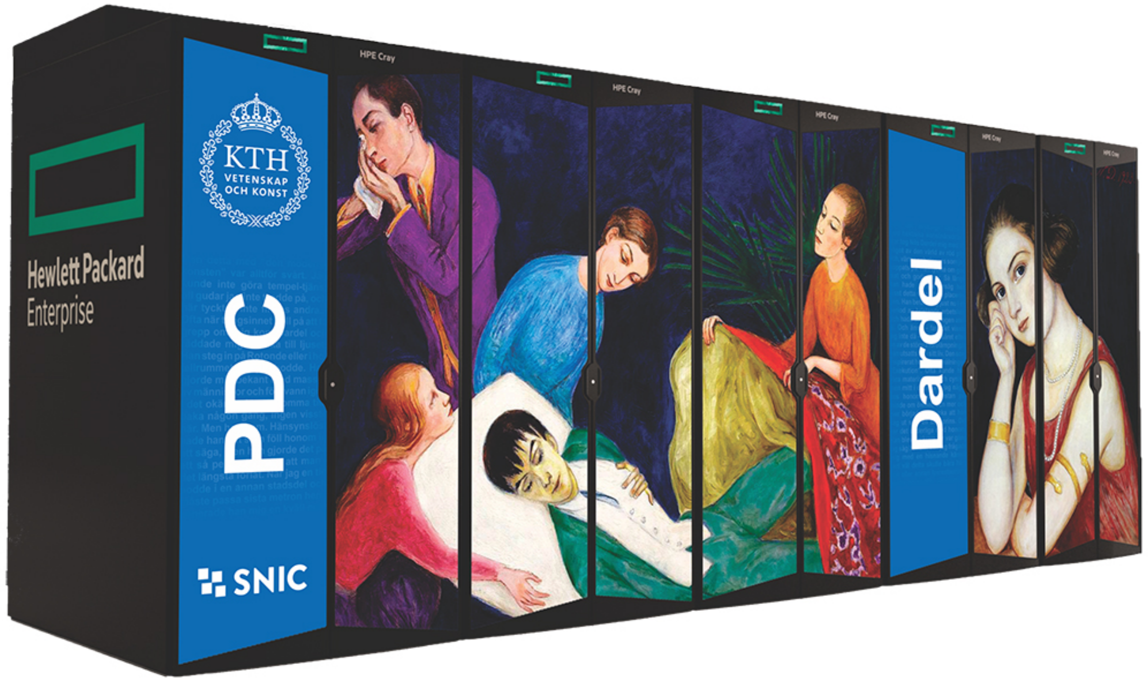
- Business partners
    - https://www.pdc.kth.se/research/business-research/pdc-partners
- A small part of Dardel nodes will be dedicated to industry/business research.
- If you are interested in purchasing HPC compute time, contact PDC Support.

Broad range of training

- Summer School
  - Introduction to HPC held every year
- SNIC Zoom-in (advertised in SNIC training newsletter)
- Workshops (see PDC events)
- University Courses that use PDC systems
  - KTH courses: DD2356, DD2365, SF2568, FDD3258, ...
  - SU courses: BL4018, BL8060

Support and System Staff

- First-Line Support
    - Provide specific assistance to PDC users related to accounts, login, allocations etc.
- System Staff
    - System managers/administrators ensure that computing and storage resources run smoothly and securely.
- Application Experts
    - Hold PhD degrees in various fields and specialize in HPC. Assist researchers in optimizing, scaling and enhancing scientific codes for current and next generation supercomputers.



Introduction to Dardel supercomputer

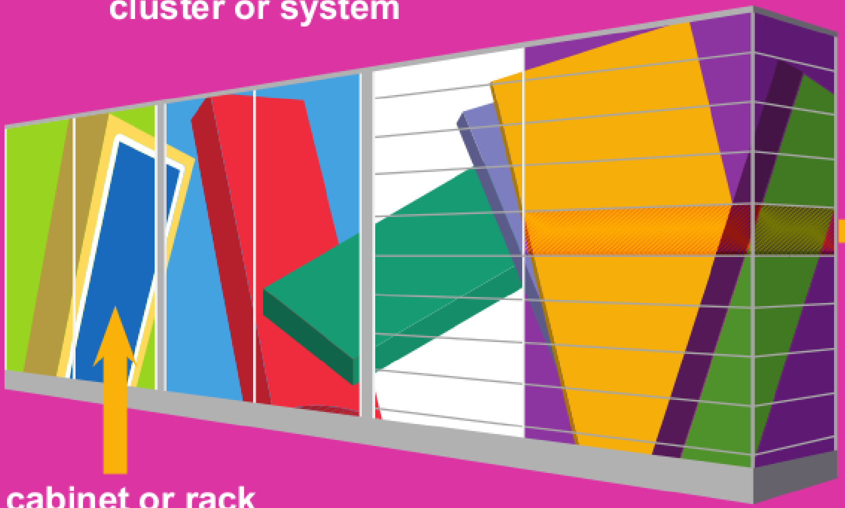
2022-10-27

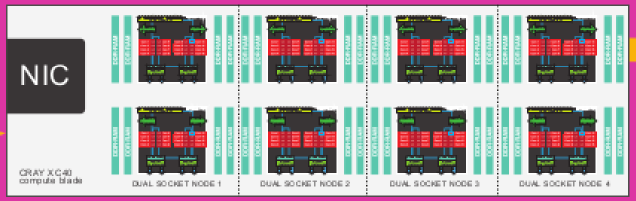
About Dardel

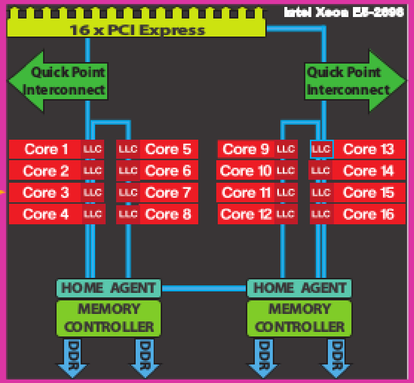
Dardel is an HPE Cray EX system

- Phase 1: CPU partition
  - 2.279 petaFlops (Top500 Nov 2021)
  - 554 CPU nodes
  - Dual AMD EPYC(TM) 64-core processors
- Phase 2: GPU partition
  - 56 GPU nodes
  - AMD EPYC(TM) processor with 64 cores
  - four AMD Instinct(TM) MI250X GPUs

Supercomputer anatomy







Cores are individual processors. Central Processing Units (CPUs) used to have just a single core, so “core” and “CPU” were used interchangeably. Now most CPU chips are multiprocessors made with multiple cores.

Supercomputer anatomy

- Dardel consists of several cabinets (also known as racks)
- Each cabinet is filled with many blades
- A single blade hosts two nodes
- A node has two AMD EPYC 7742 CPUs, each with 64 cores clocking at 2.25GHz

Compute nodes

Number of nodes	RAM per node	Name
488	256 GB	Thin nodes
20	512 GB	Large nodes
8	1 TB	Huge nodes
2	2 TB	Giant nodes
36	256 GB	Business nodes

Total: **554** Nodes (128 cores per node)

Non-uniform memory access (NUMA)

- Memory access time depends on the relative location of memory
- Accessing local memory is faster compared to non-local memory
- On Dardel compute node: 8 NUMA domains (16 cores in each NUMA domain)

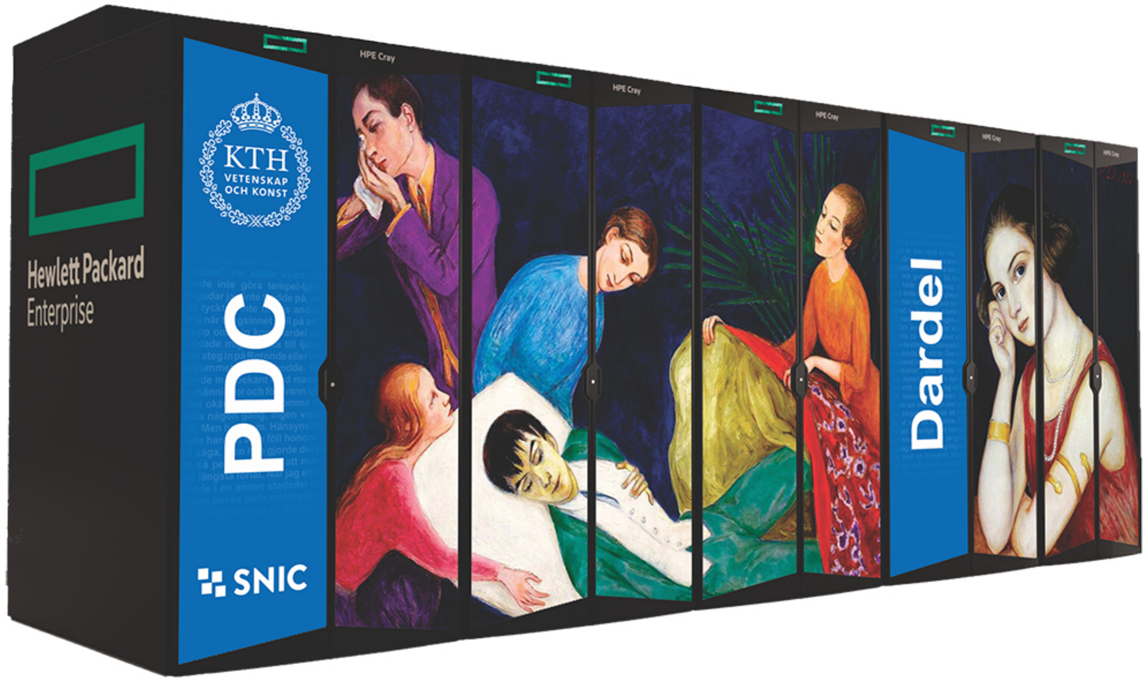
```
salloc -N 1 ...
srun -n 1 numactl --hardware
```

Non-uniform memory access (NUMA)

```
available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 128 129 130 ... 141 142 143
node 0 size: 31620 MB
node 0 free: 30673 MB
...
node 7 cpus: 112 113 114 115 116 ... 123 124 125 126 127 240 241 242 ... 253 254 255
node 7 size: 32246 MB
node 7 free: 31375 MB
```

Non-uniform memory access (NUMA)

```
node distances:
node   0   1   2   3   4   5   6   7
  0:  10  12  12  12  32  32  32  32
  1:  12  10  12  12  32  32  32  32
  2:  12  12  10  12  32  32  32  32
  3:  12  12  12  10  32  32  32  32
  4:  32  32  32  32  10  12  12  12
  5:  32  32  32  32  12  10  12  12
  6:  32  32  32  32  12  12  10  12
  7:  32  32  32  32  12  12  12  10
```



Account, Login and File System

2022-10-27

PDC support documentation

Getting a PDC account

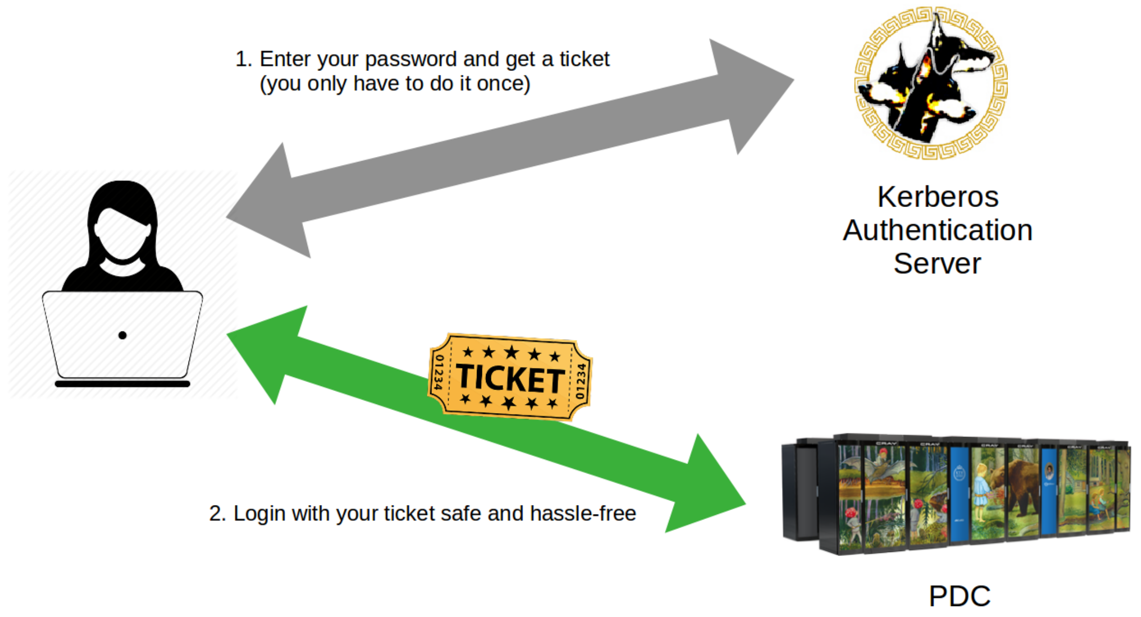
- From SUPR
    - Get a SUPR account
    - Join a project with time allocation on Dardel
        - PI: create proposal for small/medium/large allocation
        - collaborators: added to project by PI
    - Request a PDC account from SUPR
- From PDC webpage
    - Attend a course or training event
    - Fill in the form and provide a copy of passport or national ID

Time allocation (project)

- Unit: core-hours per month
- SNIC projects are managed in SUPR.
- Course/Workshop allocations are managed locally at PDC.
- Use `projinfo` to list the projects that you have access to.

Login

- Method 1: Use Kerberos ticket
    - Windows: PuTTY + NIM
    - Linux: openssh-client
    - macOS: homebrew-openssh-gssapi
- Method 2: Use SSH keys
    - Upload your public key in PDC login portal


1. Enter your password and get a ticket
(you only have to do it once)
Kerberos Authentication Server
TICKET
2. Login with your ticket safe and hassle-free
PDC

Login with Kerberos ticket

- Edit `/etc/krb5.conf` on Linux/macOS
- `kinit -f <your-username>@NADA.KTH.SE`
  - use NIM on Windows
  - use `/usr/bin/kinit -f ...` on macOS
- `ssh -o GSSAPIDelegateCredentials=yes -o GSSAPIKeyExchange=yes -o GSSAPIAuthentication=yes <your-username>@dardel.pdc.kth.se`
  - use PuTTY on Windows
  - you can save SSH options in `~/.ssh/config` on Linux/macOS

Login with SSH keys

- Generate SSH key pair

```
ssh-keygen -t ed25519 -f ~/.ssh/id-ed25519-pdc
```

  - Create `~/.ssh` folder if it doesn't exist

  ```
  mkdir ~/.ssh && chmod 700 ~/.ssh
  ```

- Upload SSH public key in PDC login portal
- Login using SSH key

```
ssh -i ~/.ssh/id-ed25519-pdc <your-username>@dardel.pdc.kth.se
```

Exercise: Log in to Dardel

File System

Lustre File System (Klemming, total size 12 PB (12,000 TB))

- Open-source massively parallel distributed file system
- Optimized for handling data from many clients
- Home directory (25 GB quota, with backup)

```
/cfs/klemming/home/[u]/[username]
```

- Project directory

```
/cfs/klemming/projects/snic/[projectname]
```

- Scratch directory

```
/cfs/klemming/scratch/[u]/[username]
```

Exercise: Home and project directories

- Home directory

  ```
  cd && pwd
  ```

  or

  ```
  echo $HOME
  ```

- Project directory

  ```
  projinfo
  ```

Find out your groups

In addition to `projinfo`, your groups also indicate the projects that you have access to.

```
groups
```

- groups starting with `cac-` are associated with compute projects
- groups starting with `pg_` are associated with storage projects

Find out your storage quota

```
projinfo
```

```
$HOME folder
Path: /cfs/klemming/home/u/user
Storage: ... GiB
Number of files: ...
```

```
Information for storage project: snicYYYY-X-XX (PI: ...)
...
Active from ... to ...
Members: ...
Max quota: ... GiB, ... files
Path: /cfs/klemming/projects/snic/...
Storage: ... TiB
Number of files: ...
```

Use of file system

- Good practice
  - Minimize the number of I/O operations
  - Avoid creating too many files
  - Avoid creating directories with a large numbers of files
- Bad practice
  - Small reads
  - Opening many files
  - Seeking within a file to read a small piece of data

Access Control Lists

To view the access for a folder:

```
getfacl -a /cfs/klemming/home/u/user/test
```

The output looks like this:

```
# file: /cfs/klemming/home/u/user/test
# owner: me
# group: users
user::rwx
group::r-x
other::---
```

Access Control Lists

To grant the access to another user (use "-R" for recursive):

```
setfacl -m u:<uid>:r-x /cfs/klemming/home/u/user/test
```

To remove the access for another user (use "-R" for recursive):

```
setfacl -x u:<uid> /cfs/klemming/home/u/user/test
```

Using Bash shell

Introduction for beginners

Tor Kjellsson Lindblom

Content

- Bash shell and basic commands
- Files and Folders
- Input/output
- Searching in text
- Processes
- File/directory permissions
- Environment variables

Bonus material for self-studying

- Finding files
- Hotkeys
- File archiving

What is a shell?

- What you get when the terminal window is open
- A "layer" (shell) around the operating system
- **Frequently used to interact with remote systems, such as Dardel**
- Multiple types of shells exist - this is about **bash shell**
- This presentation contains the very basics, mixing in some hands-on exercises.
- **No need to finish all exercises**.

Getting a bash shell

- Linux and Mac users - just open a terminal window (or login to Dardel)
- Windows users - please login to Dardel.

Your very first commands

Explore the following commands, one at a time.

```
pwd                  (print work directory)
ls                   (list files)
ls -l                (+ extra flag)
mkdir bash_tutorial (create directory)
cd bash_tutorial    (change directory)
cd ..                (cd one step above current directory)
wget  https://swcarpentry.github.io/shell-novice/data/shell-lesson-data.zip (download file from specified location)
unzip shell-lesson-data.zip
```

The last two lines downloads and extracts tutorial material we have borrowed from Software Carpentry

https://carpentries.org/

Exercise 1

```
* Explore the contents of shell-lesson-data
* Move back and forth into the subdirectories
* Move into exercise-data and copy the file
 "numbers.txt" into a new file "numbers_copy.txt"
 by typing "cp numbers.txt numbers_copy.txt"
```

verify that the new file was created.

NB! Do not use whitespace in file/folder names.

```
* Create a new directory with some name, and move this copy there
mv numbers_copy.txt your_new_folder/.
```

Some more commands

```
cp -r dir1 dir2   (copy directory, NB "-r")
man ls            (prints manual for ls)
rm file_example   (remove file_example (careful! can not be undone))
history           (prints command line history)
cat file_example  (print all contents of file)
less file_example (view contents of file in pager)
head file_example (print first lines of file)
tail file_example (print last lines of file)
```

Exercise 2

```
* Copy an existing directory
* Display "numbers_copy.txt" in some way
* Print your command line history
* Take a peek at the manual for command ls:
* Type man ls
* Press up/down keys to scroll
* Type / to start search mode
* Try to search for the flag: -F
* Type q to quit
```

Relative vs. absolute paths

You can specify a location relative to current position, or give an absolute path to it.

```
(Relative)
cd exercise_data

(Absolute)
cd /home/tkl/shell-lesson-data/exercise-data
```

Text editors

Most people are used to GUI text editors but it is often worthwhile to master at least one editor in the terminal.

List of common editors:

- nano - easiest, minimal functionality
- vi/vim - a bit more involved, but more functionality
- emacs - even a bit more involved, but a lot of functionality

Input & Output: redirect and pipes

- Programs can display something, e.g. **"echo hello world"**
- Programs can take input, e.g. **less**
- "cat numbers_copy.txt" dumps the file to *stdout*
- "cat numbers_copy.txt | less" gives the text as input to *less* (i.e. pipe)

Try it: pipes

```
# count number of logged in users
w -h | wc -l

# to list all matching commands
history | grep -w 'ls'

# print the name of the newest file in the directory (non-dot)
ls -1tF | grep -v -E '*/|@' | head -1
```

Redirects

- Like pipes, but data is sent to/from files and not processes
- Replace a file: command > file.txt
- Append to a file: command >> file.txt (be careful to **not** mix them up!)
- Redirect file as STDIN: command < file (in case program accepts STDIN only)

Try it: Redirects

```
echo Hello World > hello.txt

ls -lH >> current_dir_ls.txt

# join two files (e.g. the two above) into one
cat file1 file2 > file3

# go through file1 and replace spaces with a new line mark, then output to file2
tr -s ' ' '\n' < file1 > file2

# -or- in more readable format
cat file1 | tr -s ' ' '\n' > file2
```

Exercise 4

```
* Step into the data-shell folder
* Type history
* Type history > history.txt
* Type ls -l and then check time stamp of history.txt
* Print the last 4 lines of history.txt using the tail command
(explore the manpage if needed)
* Instead of creating an intermediate file, find a more clever
way to print the last 4 commands by piping history into tail
```

grep

This command is for searching keyword inside files.

```
grep <pattern> <filename>  # grep lines that match <pattern>
 -or-
command | grep <pattern>  # grep lines from stdin
```

Exercise 5 [grep]

```
* Go back to the data-shell directory
* Type grep rabbit exercise-data/animal-counts/animals.csv
* Try finding all occurences of the string “rabbit” using
recursive search (adding the -R flag)

grep + pipes:
* Make a pipe that displays all files ending with "pdb" in the
 data-shell directory.
```

Processes

Uptil now we only discussed files/folders.

But we also want to run **programs.**

- All running programs and commands are *processes*
- Processes have:
    - Process ID, NAME, Command line arguments
    - input and output, Return code (integer) when complete
    - Working directory, Environment variables
- These concepts bind together all UNIX programs
- To see some runnings processes, type *top*

Foreground and background processes

Foreground

- Example: *Top*
- Keyboard is connected as input, screen to output.
- Only one such process active at a time.
- Kill it: Ctrl-c

Background

- No input connected
- You can have as many as resources allow
- Add an *&* after a command to put it in background
- To kill: use *kill* or *pkill*, or do it from within *top*

Foreground and background processes [cont]

```
Example:
./my_prog.ex
./my_prog.ex 1> output.txt 2>error.txt &
```

NB: You will most likely not use Dardel like this, but it is possible to do so by logging into a compute node.

File/directory permissions

The basics

Important to set access permission on shared objects

```
Example
$ ls -l exercise_data

drwxrwxr-x 2 tkl tkl 4096 sep 16  2021 proteins
-rw-rw-r-- 1 tkl tkl   13 sep 16  2021 numbers.txt
drwxrwxr-x 2 tkl tkl 4096 sep 16  2021 animal-counts
drwxrwxr-x 2 tkl tkl 4096 sep 16  2021 creatures
drwxrwxr-x 2 tkl tkl 4096 sep 28 12:23 writing
```

Tell you who can: read, write and execute a file/list a directory

d is for directory - then 3 groups of triple fields: user, group, others

Basic file permission manipulation

```
chmod u+rwx fileA      (add read, write, execute rights to fileA for user)

chmod o+r fileA        (add read permission for others)

chmod o-wx fileA       (remove write and execectue for others)

chmod -R <perms> <dir> (recursively apply permission changes on all of dir)

chgrp group_name fileA (change group ownership)

chown -R folder        (Change owner of folder)
```

NB: On Dardel, we also use ACLs (next slide)

Access control lists (ACLs)

```
* On Lustre (Klemming) we use more advanced access permissions.

* Normal unix permission have only one owner and group.
 With ACLs, this restriction is lifted.

* ACLs are controlled via getfacl and setfacl.

* getfacl file                       (get current stage)
* setfacl -m u:<user>:r file    (Allow read access for user)
```

In many support cases we ask users to apply the last line so we can access files.

Environment variables

Defined text strings that your programs may use

```
* In the shell, these variables define your environment
* Common practice: capital letters, e.g. $HOME, $PATH, $OMP_NUM_THREADS
* List all defined variables with printenv

Try it:
* Type echo $HOME
* Type echo $HOSTNAME
* Type echo $PATH
```

Initialization and configuration

- When the shell first starts (e.g. at login) it reads shell config files.
- The config files give you power to customize your shell to your liking.
- You can always manually test things in an open shell before putting it in the config files (recommended!)

Config files are located in $HOME and are called:

- .bashrc
- .bash_profile

Initialization and configuration [cont]

Example to try

```
* Type history
* Type HISTTIMEFORMAT="%d/%m/%y %T "
* Type history
```

Bonus material

- Finding files
- Hotkeys
- File archiving

Jump to next section

Finding things [bonus slide]

Command: *find*

```
# search for pentane.pdb in current directory
find . -name pentane.pdb

# one can search more than one dir at once
find . /cfs/klemming/nobackup/u/username -name pentane.pdb
```

Exercise 5 [bonus slide]

Bonus (for interested to do later):

- On a Lustre system, *lfs find* is faster. Same syntax.
- On a workstation: *locate* may be useful. Read manual for information.
- Type find . -name animals.csv.
- Type find . -name *.pdb
- Make a pipe that counts number of files/directories in the data-shell directory.
- Count unique logged in users on Dardel.

Tip: **w** or **users** give you a list of all currently login users, many of them have several sessions open.

Tip: You may have to use **uniq, tr -s, cut -f 1 -d " "**, and **wc -l**

Hotkeys [bonus slide]

- Shortcuts
- Most important key: **tab** for autocompletion
- You should never type full filenames or command names - **tab** can complete almost anything.

Some common commands

```
TAB               (autocompletion)
Home/Ctrl-a       (move to start of command line)
End/Ctrl-e        (move to end)
up/down           (traverse command history)
Ctrl-l            (clear the screen)
Ctrl-Shift-c      (copy)
Ctrl-Shift-v      (paste)
Ctrl-r            (command history search: backwards)
```

Exercise 6 [bonus slide]

TAB autocompletion

We will here display contents of a file using its full path, but try to type as few characters as possible.

First find your absolute path to "numbers.txt"

```
* Type find $HOME -name numbers.text
Say the path was /home/tkl/shell-lesson-data/exercise-data/numbers.txt

* Type cat /home/tk and then start hitting TAB.
 Add characters when needed to reach full path.
(use your own path)
```

File archiving [bonus slide]

```
# create tar archive gzipped on the way
tar -caf archive_name.tar.gz dir/

# extract files
tar -xaf archive_name.tar.gz -C /path/to/directory
```

- *tar* is the standard tool to save many files or directories into a single archive
- Archive files may have extensions .tar, .tar.gz etc depending on compression used.
- "f" is for filename, "a" selects compression based on suffix
- With no compression, files are simply packed
- "r" will append files to end of archive, "t" will list archive
- Individual files can be compressed directly with e.g. gzip. (gzip file, gunzip file.gz)

Exercise 7 [bonus slide]

```
* Make a tar.gz archive of shell-lesson-data
* Make a tar archive, compare sizes
* List the files inside the archive
```

Building software with EasyBuild

Henric Zazzi

Link to download the slides: easybuild.pdf

Using Singularity

Henric Zazzi

Link to download the slides: singularity.pdf

Compiling and running your own code

Johan Hellsvik, Xin Li

Cray programming environment (CPE)

Reference page: Compilers and libraries

The Cray Programming Environment (CPE) provides consistent interface to multiple compilers and libraries.

- In practice, we recommend
  - `ml cpeCray/21.11`
  - `ml cpeGNU/21.11`
  - `ml cpeAMD/21.11`
- The `cpeCray`, `cpeGNU` and `cpeAMD` modules are available after `ml PDC/21.11`
- No need to `module swap` or `module unload`

Compiler wrappers

- Compiler wrappers for different programming languages
  - `cc` : C compiler wrapper
  - `CC` : C++ compiler wrapper
  - `ftn` : Fortran compiler wrapper
- The wrappers choose the required compiler version and target architecture optinons.
- Automatically link to MPI library and math libraries
  - MPI library: `cray-mpich`
  - Math libraries: `cray-libsci` and `cray-fftw`

Compile a simple MPI code

- `hello_world_mpi.f90`

```
program hello_world_mpi
include "mpif.h"
integer myrank,mysize,ierr
call MPI_Init(ierr)
call MPI_Comm_rank(MPI_COMM_WORLD,myrank,ierr)
call MPI_Comm_size(MPI_COMM_WORLD,mysize,ierr)
write(*,*) "Processor ",myrank," of ",mysize,": Hello World!"
call MPI_Finalize(ierr)
end program
```

```
ftn hello_world_mpi.f90 -o hello_world_mpi.x
```

What flags do the `ftn` wrapper activate?

- Use the flag `-craype-verbose`

  ```
  ftn -craype-verbose hello_world_mpi.f90 -o hello_world_mpi.x
  ```

Compile a simple MPI code

```
user@uan01:> srun -n 8 ./hello_world_mpi.x
 Processor           4  of           8 : Hello World!
 Processor           6  of           8 : Hello World!
 Processor           7  of           8 : Hello World!
 Processor           0  of           8 : Hello World!
 Processor           1  of           8 : Hello World!
 Processor           2  of           8 : Hello World!
 Processor           3  of           8 : Hello World!
 Processor           5  of           8 : Hello World!
```

Compile a simple linear algebra code

Link to the code

Use cray-libsci

```
ml PDC/21.11 cpeGNU/21.11
```

```
cc dgemm_test.c -o dgemm_test_craylibsci.x
```

Compile a simple linear algebra code

Use openblas

```
ml openblas/0.3.18-openmp
```

```
export OPENBLASROOT=/pdc/software/21.11/spack/spack/opt/spack/cray-sles15-zen2/gcc-11.2.0/openblas-0.3.18-2hewsuvypaots3husxzoz6ohiuixj464
```

```
cc dgemm_test.c -o dgemm_test_openblas.x -I$OPENBLASROOT/include -L$OPENBLASROOT/lib -lopenblas
```

Check the linked libraries

```
ldd dgemm_test_craylibsci.x
```

```
ldd dgemm_test_openblas.x
```

Check the linked libraries

```
ldd dgemm_test_craylibsci.x

...
libsci_gnu_82.so.5 => /opt/cray/pe/lib64/libsci_gnu_82.so.5
...
```

```
ldd dgemm_test_openblas.x

...
libopenblas.so.0 => /.../openblas-0.3.18.../lib/libopenblas.so.0
...
```

Exercise: Compile and run the dgemm_test code

- Run on a single core in the `shared` partition

```
salloc -n 1 -t 10 -p shared -A edu2210.intropdc --reservation=intropdc-2022-10-28
srun -n 1 ./dgemm_test_craylibsci.x
srun -n 1 ./dgemm_test_openblas.x
exit
```

- Expected output:

```
    2.700      4.500      6.300      8.100      9.900     11.700     13.500
    4.500      8.100     11.700     15.300     18.900     22.500     26.100
    6.300     11.700     17.100     22.500     27.900     33.300     38.700
```

Exercise: Compile and run `fftw_test` code

```
ml cray-fftw/3.3.8.12

wget https://people.math.sc.edu/Burkardt/c_src/fftw/fftw_test.c

cc --version
cc fftw_test.c -o fftw_test.x

ldd fftw_test.x

salloc -n 1 -t 10 -p shared -A edu2210.intropdc --reservation=intropdc-2022-10-28
srun -n 1 ./fftw_test.x
```

Compilation of large program

- Compilation of NWChem
- Compilation of VASP
- Compilation of VeloxChem
- Compilation of DFTD4

Environment variables for manual installation of software

- Environment variables for compilers

```
export CC=cc
export CXX=CC
export FC=ftn
export F77=ftn
```

- Environment variables for compiler flags
  - add `-I`, `-L`, `-l`, etc. to Makefile
- Environment variables at runtime
  - prepend to `PATH`, `LD_LIBRARY_PATH`, etc.

What happens when loading a module

```
ml show nwchem/7.0.2
```

```
whatis("NWChem: Open Source High-Performance Computational Chemistry")
whatis("Homepage: https://www.nwchem-sw.org/")
whatis("URL: https://www.nwchem-sw.org/")
conflict("nwchem")
setenv("NWCHEM_TOP","/pdc/software/21.11/other/nwchem/7.0.2")
setenv("NWCHEM_BASIS_LIBRARY","/pdc/software/21.11/other/nwchem/7.0.2/data/libraries/")
setenv("NWCHEM_NWPW_LIBRARY","/pdc/software/21.11/other/nwchem/7.0.2/data/libraryps/")
setenv("DEFAULT_NWCHEMRC","/pdc/software/21.11/other/nwchem/7.0.2/data/default.nwchemrc")
prepend_path("PATH","/pdc/software/21.11/other/nwchem/7.0.2/bin")
```

What happens when loading a module

```
ml show dftd4/3.3.0
```

```
whatis("Generally Applicable Atomic-Charge Dependent London Dispersion Correction")
whatis("Homepage: https://github.com/dftd4/dftd4")
whatis("URL: https://github.com/dftd4/dftd4")
conflict("dftd4")
prepend_path("CMAKE_PREFIX_PATH","/pdc/software/21.11/other/dftd4/3.3.0")
prepend_path("LD_LIBRARY_PATH","/pdc/software/21.11/other/dftd4/3.3.0/lib64")
prepend_path("LIBRARY_PATH","/pdc/software/21.11/other/dftd4/3.3.0/lib64")
prepend_path("PATH","/pdc/software/21.11/other/dftd4/3.3.0/bin")
```

When running your own code

- Load correct programming environment (e.g. `cpeGNU`)
- Load correct dependencies (e.g. `openblas` if your code depends on it)
- Properly prepend to environment variables (e.g. `PATH`, `LD_LIBRARY_PATH`)
- Choose correct SLURM settings

SLURM settings for hybrid MPI/OpenMP code

- `--nodes` number of nodes
- `--ntasks-per-node` number of MPI processes
- `--cpus-per-task` 2 x number of OpenMP threads (because of SMT)
- `OMP_NUM_THREADS` number of OpenMP threads
- `OMP_PLACES` cores

Example job script

- 64 MPI x 2 OMP per node (main parition)

```
#!/bin/bash

#SBATCH -A ...
#SBATCH -J my_job
#SBATCH -t 01:00:00
#SBATCH -p main

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=64
#SBATCH --cpus-per-task=4

module load ...

export OMP_NUM_THREADS=2
export OMP_PLACES=cores

srun ...
```

Example job script

- 2 MPI x 2 OMP per node (shared partition)

```
#!/bin/bash

#SBATCH -A ...
#SBATCH -J my_job
#SBATCH -t 01:00:00
#SBATCH -p shared

#SBATCH --ntasks=2
#SBATCH --cpus-per-task=4

module load ...

export OMP_NUM_THREADS=2
export OMP_PLACES=cores

srun ...
```

Exercise: Hybrid MPI/OpenMP code for matrix-matrix multiplication

- Preparation

```
mkdir -p matmul_test && cd matmul_test
```

```
ml PDC/21.11
ml numpy/1.20.3-gcc11.2-py38
```

- If you are interested in how mpi4py and numpy were compiled, see this page

Exercise: Hybrid MPI/OpenMP code for matrix-matrix multiplication

- Copy python code matmul_mpi_omp_test.py to the same folder
- Copy job script job-n2.sh
    - for running on 2 MPI processes with different number of OpenMP threads
- Copy job script job-n4.sh
    - for running on 4 MPI processes with different number of OpenMP threads
- Submit two jobs

Exercise: Hybrid MPI/OpenMP code for matrix-matrix multiplication

- Result

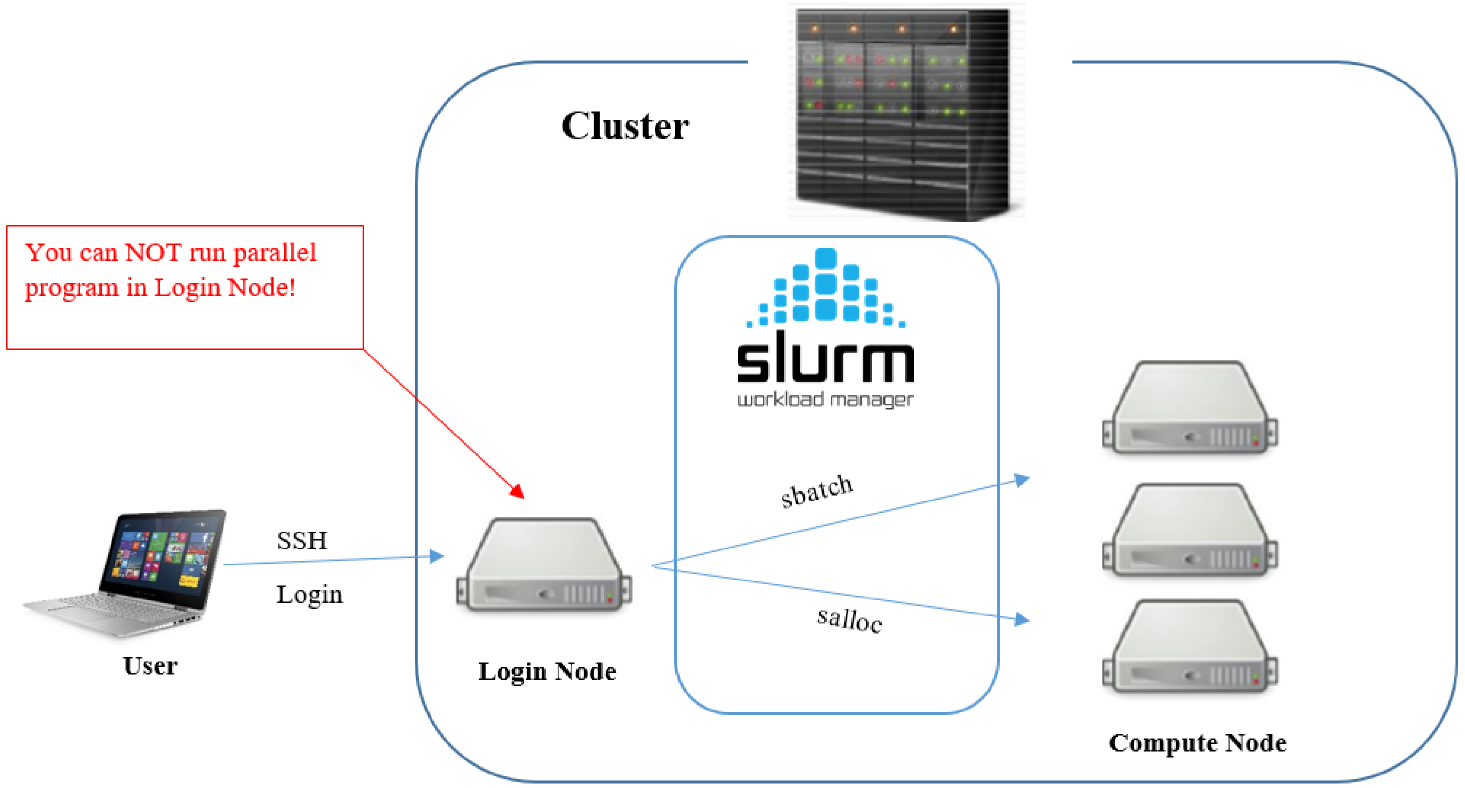
Setting	Timing
2 MPI x 8 OMP	Time spent in matmul: 1.714 sec
2 MPI x 4 OMP	Time spent in matmul: 3.110 sec
2 MPI x 2 OMP	Time spent in matmul: 5.273 sec
4 MPI x 4 OMP	Time spent in matmul: 1.698 sec
4 MPI x 2 OMP	Time spent in matmul: 2.747 sec
4 MPI x 1 OMP	Time spent in matmul: 4.559 sec

Running jobs with efficient utilization of hardware

Xavier Aguilar

How we run jobs on a Supercomputer

- Edit files
- Compile programs
- Run light tasks
- Submit jobs
- **Do not** run parallel jobs on the login node!



What is SLURM ?

Simple Linux Utility for Resource Management

- Open source, fault-tolerant, and highly scalable cluster management and job scheduling system
    - Allocates access to resources
    - Provides a framework for work monitoring on the set of allocated nodes
    - Arbitrates contention for resources

Types of jobs?

- **Batch jobs**
  - the user writes a job script indicating the number of nodes, cores, time needed, etc.
  - the script is submitted to the batch queue
  - The user retrieves the output files once the job is finished
- **Interactive jobs:**
  - the user runs a command that allocates interactive resources on a number of cores
  - this creates an interactive job that awaits in the queue as any other job
  - when the job reaches the front of the queue, the user gets access to the resources and can run commands there

How jobs are scheduled?

- **Age**: the time the job has been in the queue
- **Job size**: number of nodes or cores requested
- **Partition**: a factor associated with each node partition
- **Fair-share**: the difference between the computing resources promissed and the amount of resources computed

SLURM basic commands

Submit a job to the queue:

```
sbatch <script>
```

List queued/running jobs belonging to a user:

```
squeue -u <username>
```

Cancel a job:

```
scancel <job-id>
```

Get information on partitions and nodes

```
sinfo
```

Job scripts (pure MPI)

```
#!/bin/bash -l
# The -l above is required to get the full environment with modules

# Set the allocation to be charged for this job
# not required if you have set a default allocation
#SBATCH -A snicYYYY-X-XX

# The name of the script is myjob
#SBATCH -J myjob

# The partition
#SBATCH -p main

# 10 hours wall-clock time will be given to this job
#SBATCH -t 10:00:00

# Number of nodes
#SBATCH --nodes=4

# Number of MPI processes per node
#SBATCH --ntasks-per-node=128

# Loading modules needed by your job
module add X
module add Y

# Run the executable named myexe
# and write the output into my_output_file
srun ./myexe > my_output_file
```

Partitions

- Nodes are logically grouped into partitions
- There are four partitions that can be used on Dardel
    - **main**: Thin nodes (256 GB RAM), whole nodes, maximum 24 hours job time
    - **long**: Thin nodes (256 GB RAM), whole nodes, maximum 7 days job time
    - **shared**: Thin nodes (256 GB RAM), job shares node with other jobs, maximum 24 hours job time
    - **memory**: Large/Huge/Giant nodes (512 Gb - 2 TB RAM), whole nodes, 24 hours job time

Job scripts (MPI + OpenMP)

```
#!/bin/bash -l
# The -l above is required to get the full environment with modules

# Set the allocation to be charged for this job
#SBATCH -A snicYYYY-X-XX

# The name of the script is myjob
#SBATCH -J myjob

# The partition
#SBATCH -p main

# 10 hours wall-clock time will be given to this job
#SBATCH -t 10:00:00

# Number of Nodes
#SBATCH --nodes=4

# Number of MPI tasks per node
#SBATCH --ntasks-per-node=16
# Number of logical cores hosting OpenMP threads. Note that cpus-per-task is set as 2x OMP_NUM_THREADS
#SBATCH --cpus-per-task=16

# Number and placement of OpenMP threads
export OMP_NUM_THREADS=8
export OMP_PLACES=cores

# Run the executable named myexe
srun ./myexe > my_output_file
```

Exercise 1: Basic SLURM commands

- In this exercise we are going to test some basic SLURM commands.
- You will:
  - Create and compile a simple MPI program
  - Submit it to the queues
  - Inspect the queues
  - Inspect the partitions
  - Check the output of the job

MPI Hello world

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
  int world_size,world_rank;

  // Initialize the MPI environment
  MPI_Init(NULL, NULL);
  // Get the number of processes
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);
  // Get the rank of the process
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  // Print off a hello world message
  printf("Hello world from rank %d out of %d process\n", world_rank, world_size);
  // Finalize the MPI environment.
  MPI_Finalize();
  return 0;
  }
```

Exercise 1

- You can find the code from the previous slide here
- Save the file on Dardel, compile the code and generate a binary called *hello_mpi*

Exercise 1

- Take the job script that you can find here and modify it accordingly to:
  - Use the proper allocation required, for this course it is *edu2210.intropdc*
  - Use one node for the job
  - Use 4 cores from that node

Exercise 1

- Submit this script using **sbatch**
- Check the queue using **squeue -u <your_username>**
    - What's the ID of the job?
    - Is it already running? If so, which node was allocated for the job?
- Once the job finishes check the job output. Where is it saved?

Note:

Notice that we run our program with just:

```
srun ./hello_mpi
```

It would be also possible to run our program with:

```
srun -N 1 -n 4 ./hello_mpi
```

However, we don't need to specify those flags because SLURM takes the *-N* and *-n* values from the *BATCH* directives in the script

Exercise 1

- Use **sinfo** to check the partitions
  - How many different partitions are defined? What are their names?
  - What's the partition with the highest number of nodes?
  - Name 1-2 nodes included in that same partition

Interactive jobs

Request an interactive allocation

```
salloc -A <allocation> -t <d-hh:mm:ss> -p <partition> -N <nodes> -n <cores>
```

Once the allocation is granted, a new terminal session starts (typing exit will stop the interactive session)

```
srun -n <number-of-processes> ./mybinary.x
```

It is also possible to ssh into one of the allocated nodes.

Interactive jobs

We can check what nodes have been granted with:

```
squeue -u $USER
```

or inspecting the environment variable:

```
SLURM_NODELIST
```

SLURM advanced commands

Get detailed information of a running job:

```
sstat --jobs=<your_job-id>
```

Filter the information provided by sstat

```
sstat --jobs=your_job-id --format=JobID,aveCPU,MaxRRS,NTasks
```

Tip: Use *sstat -e* to see all possible fields for the *format* flag

SLURM advanced commands

To get information on past jobs:

```
sacct
```

Get detailed information of a certain job:

```
sacct --jobs=<your_job-id> --starttime=YYYY-MM-DD
```

```
sacct --starttime=2019-06-23 --format=JobName,CPUTime,NNodes,MaxRSS,Elapsed
```

SLURM advanced commands

Quick performance summary for a finished job:

```
seff <jobid>
```

Common reasons for inefficient jobs

- Not all nodes allocated are used
- Not all cores within a node are used (if it's not intentional)
- Many more cores than the available are used
- Inneficient use of the file system
- Using the wrong partition

Exercise 2

- In this exercise we are going to use some more advanced SLURM commands to explore job performance.
- You will:
    - Create and compile a simple MPI program
    - Submit it to the queues
    - Check job data using sacct

Exercise 2

- You can find the sample code for this exercise here
- Compile the code and generate a binary called *vector_mpi*

Exercise 2

- You can find the job script for this exercise here
- Save the script on Dardel and submit the job. Once the job finishes check its output.
- Inspect the job performance data using **sacct**
    - Use: -j <job-id> --format=JobID,JobName,Elapsed,ReqMem,MaxRSS

      Tip: use flag "--unit=M" to see memory units in MB

Exercise 2

- Use the **seff <job-id>** command for quick job efficiency overview.

```
Job ID: 211499
Cluster: dardel
User/Group: xaguilar/xaguilar
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 8
CPU Utilized: 00:00:01
CPU Efficiency: 0.37% of 00:04:32 core-walltime
Job Wall-clock time: 00:00:34
Memory Utilized: 549.25 MB (estimated maximum)
Memory Efficiency: 7.73% of 6.94 GB (888.00 MB/core)
```

Job arrays

When we have several similar jobs that can be packed within a single job

```
#!/bin/bash -l
#SBATCH -A project
#SBATCH -N 1
#SBATCH -t 00:10:00

for i in $(seq 0 9); do
        srun -n 1 myprog $i
done
```

Job arrays

```
#!/bin/bash -l
#SBATCH -A project
#SBATCH -N 1
#SBATCH -t 00:01:00
#SBATCH -a 0-9

srun -n 1 myprog $SLURM_ARRAY_TASK_ID
```

$SLURM_ARRAY_TASK_ID contains a number 0-9 identifying each job in the array (defined with -a)

Note: Too many short jobs is bad for performance and can slow down the whole queue system. Ideally each job should be at least 10 minutes long.

Job arrays

Once submitted, the job array gets a job id assigned that can be used to manipulate all jobs at once:

```
> sbatch job_array.sh
Submitted batch job 6975769
> scancel 6975769
```

For more info on job arrays check the SLURM website:
https://slurm.schedmd.com/job_array.html

Packing short jobs

Several short jobs can be packed together into a single job where they run serially

```
#!/bin/bash -l
#SBATCH -A project
#SBATCH -N 1
#SBATCH -t 00:10:00

for arg in "$@"; do
        srun -n 1 myprog $arg
done
```

The job is submitted with:

```
sbatch packed_job.sh x0 x1 x2 x3 x4 x5 x6 x7 x8 x9
```

Using fewer cores

Reduce the number of cores used per job and run multiple instances of the job in a single node

```
#!/bin/bash -l
#SBATCH -A project
#SBATCH -N 1
#SBATCH -t 00:01:00

export OMP_NUM_THREADS=32

srun myprog $1
```

Using fewer cores

```
#!/bin/bash -l
#SBATCH -A project
#SBATCH -N 1
#SBATCH -t 00:08:00

export OMP_NUM_THREADS=4

srun ./inner.sh "$@"
```

```
#!/bin/bash

for arg in "$@"; do
        myprog $arg &
done

wait
```

Good SLURM practices

- Avoid too many short jobs
- Avoid massive output to STDOUT
- Try to provide a good estimate of the job duration before submitting

Using MATLAB

Johan Hellsvik

Using MATLAB

MATLAB is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks.

MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.

MATLAB at PDC

Licensing

In order to use MATLAB you need to have a MATLAB license.

KTH has a MATLAB site license valid for academic users that are affiliated with KTH or with other universities. The license includes all MATLAB toolboxes.

To access the installation of MATLAB on Dardel, please contact PDC support and request access.

Interactive and batch use of MATLAB

MATLAB can be run on Dardel both in interactive sessions, with or without a graphical user interface, or in batch jobs.

Running interactively

Matlab can be run interactively on allocated cores. To book 24 cores for one hour

```
salloc -n 24 -t 1:00:00 -p shared -A edu2210.intropdc
```

Wait for cores to be reserved

```
salloc: Granted job allocation 591571
salloc: Waiting for resource configuration
salloc: Nodes nid001015 are ready for job
```

Log in to the node where the cores reside

```
ssh -X nid00105
```

and start MATLAB with graphical user interface (GUI)

```
ml PDC/21.11
ml matlab/r2022b
matlab
```

if you have not got X11 forwarding active, start MATLAB without GUI

```
ml PDC/21.11
ml matlab/r2022b
matlab -nodisplay -nodesktop -nosplash
```

Parallel computation with Matlab

MATLAB implements a large set of solutions for parallel computing.

- Vectorization
- Automated parallel computing support
- Parallel execution of loops with `parfor`
- Running multiple serial batch jobs

Parallel execution with `parfor`

Calculation of the spectral radius of a matrix

Consider first serial code: spectral_serial.m

```
tic
disp('Start serial calculation of spectral radius')
n = 500;
A = 1000;
a = zeros(n);
for i = 1:n
    a(i) = max(abs(eig(rand(A))));
end
toc
disp('End serial calculation of spectral radius')
```

Execution time on one core: 297 s

Can this be parallelized? How large is the overhead?

A `parfor` loop can be used to execute loop iterations in parallel on workers in a parallel pool.

Replace `for` statements in serial code with `parfor` to obtain spectral_parfor.m

```
tic
disp('Start parallel calculation of spectral radius')
ticBytes(gcp);
n = 500;
A = 1000;
a = zeros(n);
parfor i = 1:n
    a(i) = max(abs(eig(rand(A))));
end
tocBytes(gcp)
disp('End parallel calculation of spectral radius')
toc
```

Execution time on 24 cores: 30 s

Speedup as compared to execution time on 1 core: 9.9

Parallel execution of MATLAB in batch jobs

Also for batch job use of MATLAB, consider to use the shared partition of Dardel. Example job script jobMATLABn16.sh

```
#!/bin/bash -l
#SBATCH -A snicYYYY-X-XX
#SBATCH -J myjob
#SBATCH -p shared
#SBATCH -n 16
#SBATCH -t 01:00:00

# Load the Matlab module
ml add PDC/21.11
ml matlab/r2022b

# Run matlab taking your_matlab_program.m as input
matlab -nodisplay -nodesktop -nosplash < your_matlab_program.m > your_matlab_program.out
```

Running multiple serial batch jobs

```
#!/bin/bash -l
#SBATCH -A snicYYYY-X-XX
#SBATCH -J myjob
#SBATCH -p shared
#SBATCH -n 24
#SBATCH -t 02:00:00

# Load the Matlab module
ml add PDC/21.11
ml matlab/r2022b

# Run matlab for 24 individual programs serial_program_1.m,
# serial_program_2.m ... and print output in files logfile_1, logfile_2, ...
# The input files must be in the directory where you submit this script.
# This is also where the output will be created.

for i in {1..24} ; do
    matlab -nosplash -nodesktop -nodisplay < serial_program_${i}.m > logfile_$i &
done
# this wait command must be present since otherwise the job will terminate immediately
wait
```

Exercise 1

Calculating spectral radius in interactive session

- Request node / cores for interactive session
- Log in to the node, and start MATLAB
- Experiment with spectral_serial.m and spectral_parfor.m
    - Compare runtimes serial code vs parallel code for n = 200; A = 500;
    - Change values of n and A. How does relative runtime change?

Exercise 2

Calculating spectral radius in batch job

- Edit the template script jobMATLABn16.sh. You will need to
  - Specify allocation and reservation
  - Change the name of the MATLAB program
  - Consider to change the time for the job and number or cores
- Launch jobs for both the serial and the parallel versions of the spectral radius code

Using Python virtual environment

Xin Li

Why virtual environment?

- We often need to use a number of Python packages

```
import A
from B import C
import D as E
```

- When working with multiple projects, it is not uncommon that different projects have conflicting requirements of packages
- On HPC platform, different users may have conflicting needs of packages

Why virtual environment?

Without virtual environment, Python packages are installed

- either in system site directory

```
python3 -c 'import site; print(site.getsitepackages())'
```

- or in the so-called user base

```
echo $HOME/.local
```

Exercise: Check python site directory

```
which python3
python3 -c 'import site; print(site.getsitepackages())'
```

```
ml cray-python/3.9.4.2
which python3
python3 -c 'import site; print(site.getsitepackages())'
```

```
ml PDC/21.11 Anaconda3/2021.05
which python3
python3 -c 'import site; print(site.getsitepackages())'
```

Python virtual environment

- Isolated run-time environment
- Install and execute Python packages without interfering with the outside world
- Two ways of creating and managing virtual environment
    - `venv`
    - `conda`

Virtual environment with `venv`

- Recommendation: use with cray-python

```
ml cray-python/3.9.4.2

cd $HOME
python3 -m venv myenv

source myenv/bin/activate
```

```
(myvenv) user@uan01:~>
```

Virtual environment with `venv`

- Check site-packages directory

```
(myenv) user@uan01:~> python3 -c 'import site; print(site.getsitepackages())'
['/cfs/klemming/home/u/user/myenv/lib/python3.9/site-packages']
```

- Install a Python package

```
(myenv) user@uan01:~> python3 -m pip install yapf

(myenv) user@uan01:~> which yapf
/cfs/klemming/home/u/user/myenv/bin/yapf
```

Virtual environment with `venv`

- Deactivate a virtual environment

```
(myenv) user@uan01:~> deactivate
user@uan01:~>
```

Virtual environment with `conda`

- Load Anaconda3

```
ml PDC/21.11 Anaconda3/2021.05
```

- Initialize conda

```
source conda_init_bash.sh
```

```
(base) user@uan01:~>
```

Virtual environment with `conda`

- Content of `conda_init_bash.sh`

```
# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/pdc/software/21.11/eb/software/Anaconda3/2021.05/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
else
    if [ -f "/pdc/software/21.11/eb/software/Anaconda3/2021.05/etc/profile.d/conda.sh" ]; then
        . "/pdc/software/21.11/eb/software/Anaconda3/2021.05/etc/profile.d/conda.sh"
    else
        export PATH="/pdc/software/21.11/eb/software/Anaconda3/2021.05/bin:$PATH"
    fi
fi
unset __conda_setup
# <<< conda initialize <<<
```

- `conda init bash` may append the above script to your `~/.bashrc` but this is not recommended on an HPC system.

Virtual environment with `conda`

```
(base) user@uan01:~> conda create --name my-conda-env

(base) user@uan01:~> conda activate my-conda-env

(my-conda-env) user@uan01:~>
```

Virtual environment with `conda`

- Now try the site-packages directory again

```
(my-conda-env) user@uan01:~> python3 -c 'import site; print(site.getsitepackages())'
```

- Why is `site-packages` still under `Anaconda3`?

```
['/pdc/software/21.11/eb/software/Anaconda3/2021.05/lib/python3.8/site-packages']
```

Virtual environment with `conda`

```
(my-conda-env) user@uan01:~> conda install python=3.8

(my-conda-env) user@uan01:~> python3 -c 'import site; print(site.getsitepackages())'
['/cfs/klemming/home/u/user/.conda/envs/my-conda-env/lib/python3.8/site-packages']
```

Virtual environment with `conda`

```
(my-conda-env) user@uan01:~> conda deactivate
(base) user@uan01:~>
```