

# 3. HPC: Further aspects

Peter Kjellström

National Supercomputer Centre (NSC)

# Software: Shared software

On Triolith, there is a large collection of pre-installed software available for users.

Q. Why is there shared software on Triolith?

- Allows many users to run popular software
- Means users do not have to install their own software
- Software can be optimized for Triolith and managed by experts
- Licensing

In summary: pre-installed software is there to make life easier for users.

# Software: Shared software

Q. What types of software are shared?

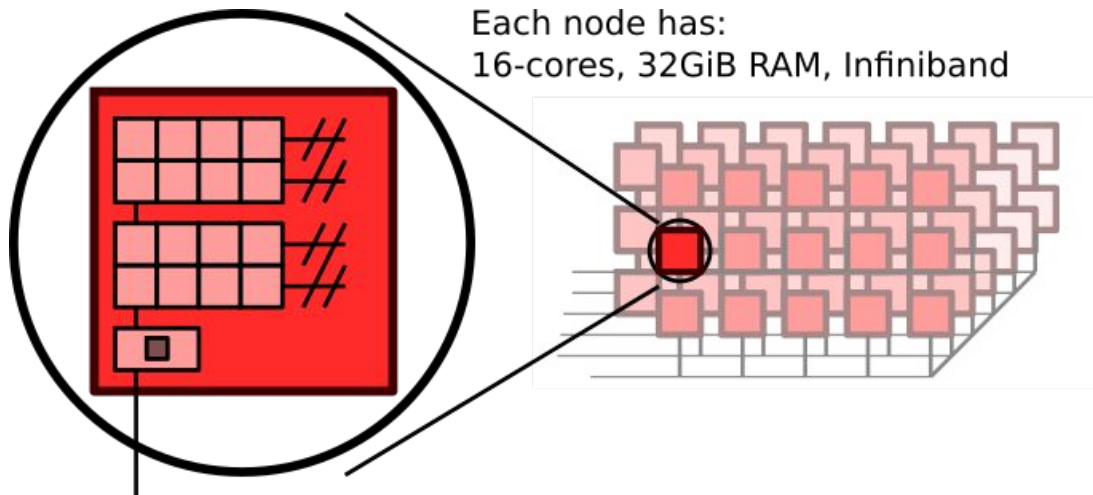
- Ready to use software
  - models for batch submission (e.g. VASP, Gaussian, Gromacs, ...)
  - utility programs and interpreted languages (e.g. R, python, perl, Matlab ...)
- Tools to build other software
  - compilers
  - build environments
  - supporting libraries

In summary: a wide variety of software is available to use.

# The parallel challenge

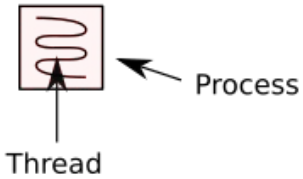
```
a = load_data(inputfiles)
for element in a:
    process(element)
store_results(ouputfiles)
```

??!

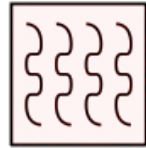


# Software: Forms of computational program

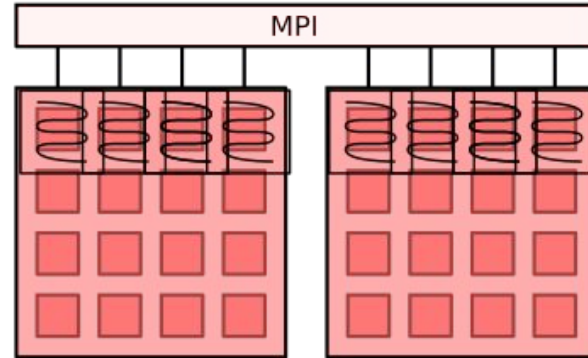
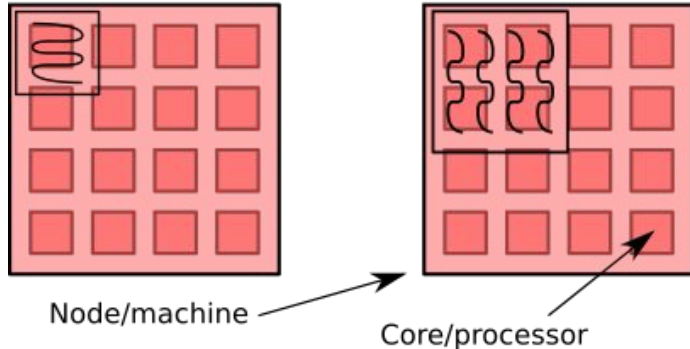
Serial program



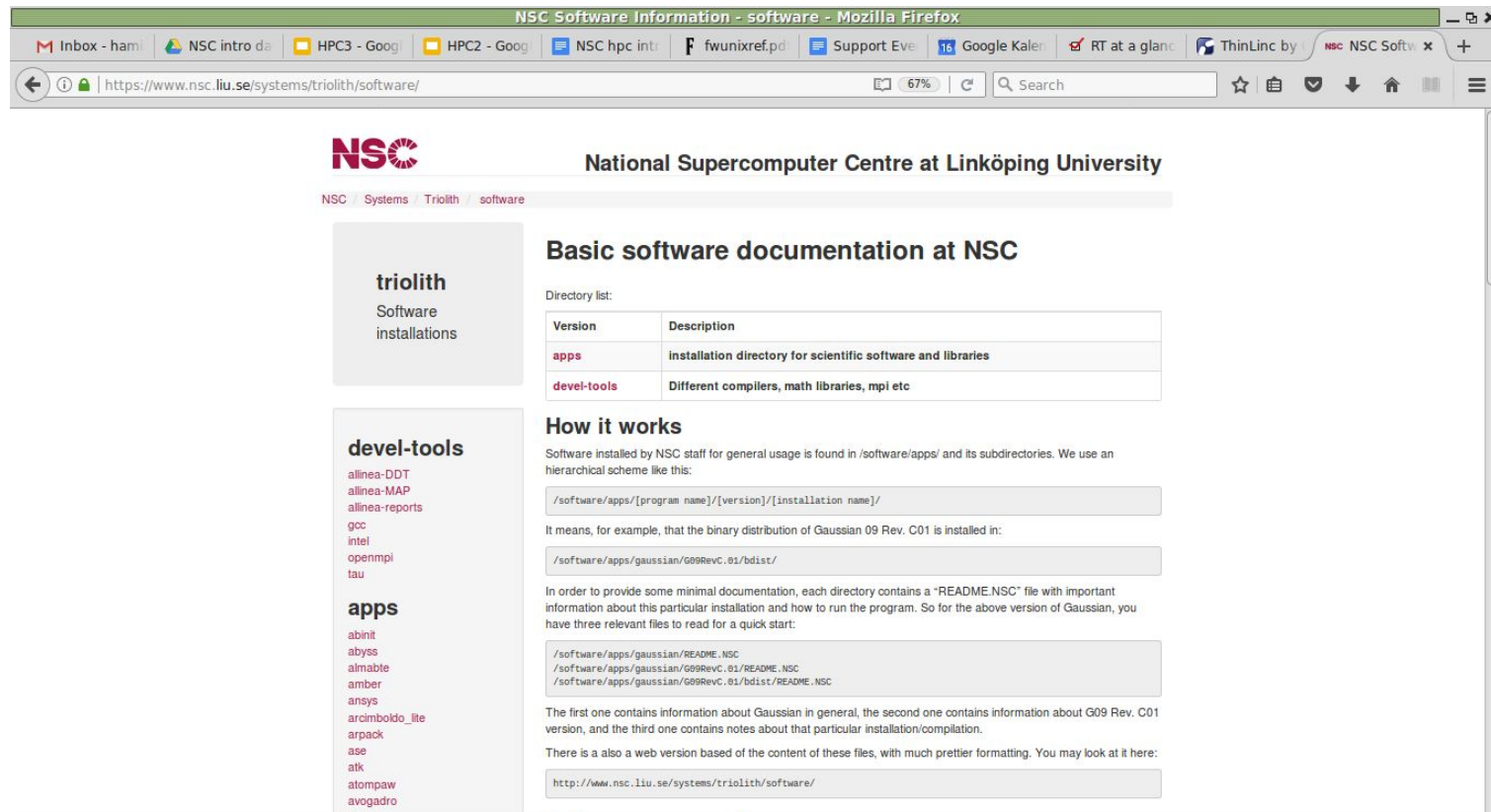
Shared memory program  
(threaded)



Distributed memory program  
(MPI)



# Software: Triolith installed software



The screenshot shows a Mozilla Firefox browser window with the address bar displaying `https://www.nsc.liu.se/systems/triolith/software/`. The page content includes the NSC logo and the title "National Supercomputer Centre at Linköping University". A breadcrumb trail shows "NSC / Systems / Triolith / software".

**triolith**  
Software installations

**devel-tools**  
allinea-DDT  
allinea-MAP  
allinea-reports  
gcc  
intel  
openmpi  
tau

**apps**  
abinit  
abyss  
almabte  
amber  
ansys  
arcimboldo\_lite  
arpack  
ase  
atk  
atompaw  
avogadro

## Basic software documentation at NSC

Directory list:

| Version            | Description  |
|--------------------|--|
| <b>apps</b>        | installation directory for scientific software and libraries |
| <b>devel-tools</b> | Different compilers, math libraries, mpi etc                 |

## How it works

Software installed by NSC staff for general usage is found in `/software/apps/` and its subdirectories. We use an hierarchical scheme like this:

```
/software/apps/[program name]/[version]/[installation name]/
```

It means, for example, that the binary distribution of Gaussian 09 Rev. C01 is installed in:

```
/software/apps/gaussian/G09RevC.01/bdist/
```

In order to provide some minimal documentation, each directory contains a "README.NSC" file with important information about this particular installation and how to run the program. So for the above version of Gaussian, you have three relevant files to read for a quick start:

```
/software/apps/gaussian/README.NSC  
/software/apps/gaussian/G09RevC.01/README.NSC  
/software/apps/gaussian/G09RevC.01/bdist/README.NSC
```

The first one contains information about Gaussian in general, the second one contains information about G09 Rev. C01 version, and the third one contains notes about that particular installation/compilation.

There is also a web version based on the content of these files, with much prettier formatting. You may look at it here:

```
http://www.nsc.liu.se/systems/triolith/software/
```

# Software: Module system, basic commands

Installed software is managed using the **module** system. Some basic module commands

|                             |  |
|-----------------------------|--|
| <code>module --help</code>  | General help with module commands                              |
| <code>module avail</code>   | List the available modules and recommendations                 |
| <code>module add ...</code> | Load the selected modules into your session                    |
| <code>module list</code>    | List your currently loaded modules (will be flushed at logout) |
| <code>module rm ...</code>  | Remove selected modules from your session                      |

# Software: Module system example

```
struthers@triolith1:~  
[struthers@triolith1 ~]$ module avail vasp  
/software/modulefiles/triolith:  
vasp/5.2.12-11Nov11  
vasp/5.2.12-11Nov11-SNIC  
vasp/5.3.2-13Sep12  
vasp/5.3.3-18Dec12  
vasp/5.3.5-01Apr14  
vasp/5.3.5-31Mar14  
vasp/5.4.1-05Feb16  
vasp/5.4.1-24Jun15  
vasp/5.4.4-18Apr17  
vasp/omp-beta-13Mar17  
vasp/recommendation(default)  
vasp-vtst/5.3.12-13Sep12+3.0a  
vasp-vtst/5.3.2-13Sep12+3.0a  
vasp-vtst/5.3.3-18Dec12+3.0c  
vasp-vtst/recommendation(default)  
vasptools/0.1  
vasptools/0.2  
vasptools/0.3  
vasptools/recommendation(default)  
[struthers@triolith1 ~]$ module load vasp/5.4.4-18Apr17  
[struthers@triolith1 ~]$
```



# Software: Cautionary note

NSC tries to provide tested and optimized software, *however* it is never perfectly tested and it will always be imperfect. The only thing you know about any piece of software is that it will contain bugs...

- Job crashes
- Job produces garbage output
- Job wastes resources by running very inefficiently

Run some short test cases to confirm your configuration is sensible.

Try to understand the basic scaling properties of the application before running any large jobs.

# Software: How do I get the software I want?

1. Check module system (`module avail`)
2. Check the installed software webpage
3. Look in `/software/apps` on Triolith
4. Build and install it yourself

## NSC software installation policy:

- Users are encouraged to install software in their `/home` or `/proj` folders
- NSC can help to install software on request
  - Global installation depends on wide or not usage

Triolith installed software: <https://www.nsc.liu.se/systems/triolith/software/>

NSC installation policy: <https://www.nsc.liu.se/software/installation-policy/>

# Software: Compilers and libraries

If you are compiling your own software, NSC recommends to load a **build environment** for ease of use

- Compilers
  - Intel: icc, ifort, impi
  - Gcc: gcc, gfortran
- MPI libraries
  - Intel (impi), OpenMPI
- Build environments:
  - e.g. `buildenv-intel/2015-1`
- Math libraries:
  - e.g. MKL

Compilers: <https://www.nsc.liu.se/software/compilers/>

MPI libraries: <https://www.nsc.liu.se/software/mpi-libraries/>

Build environments: <https://www.nsc.liu.se/software/buildenv/>

Math libraries: <https://www.nsc.liu.se/software/math-libraries/>

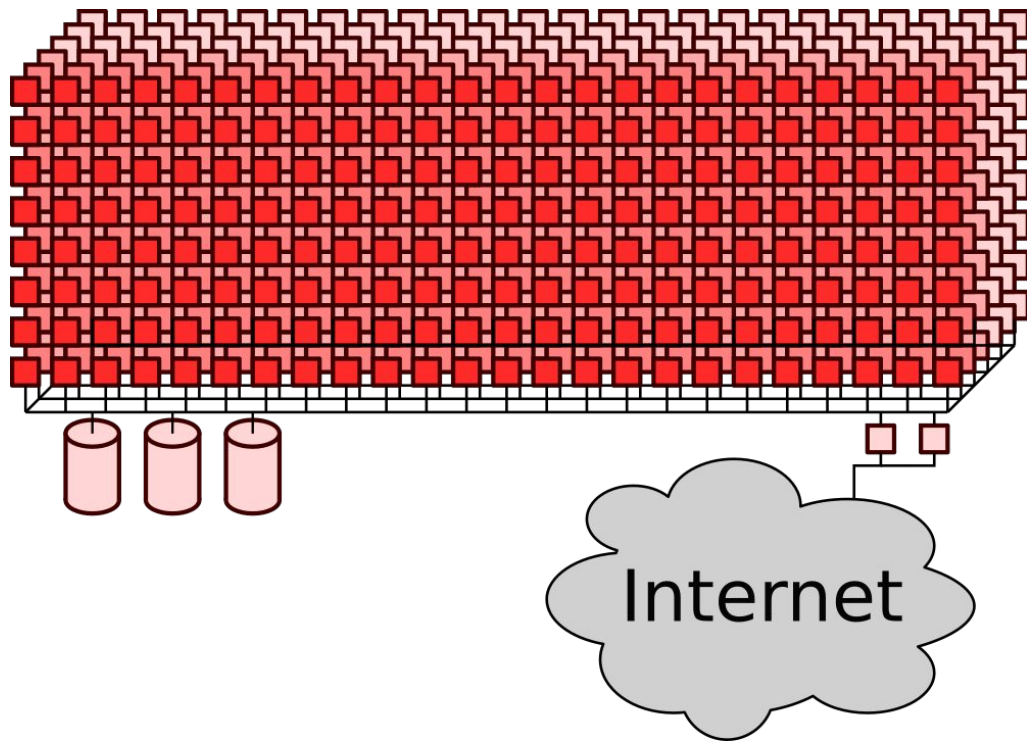
# The utilization challenge

~1000 active users per year

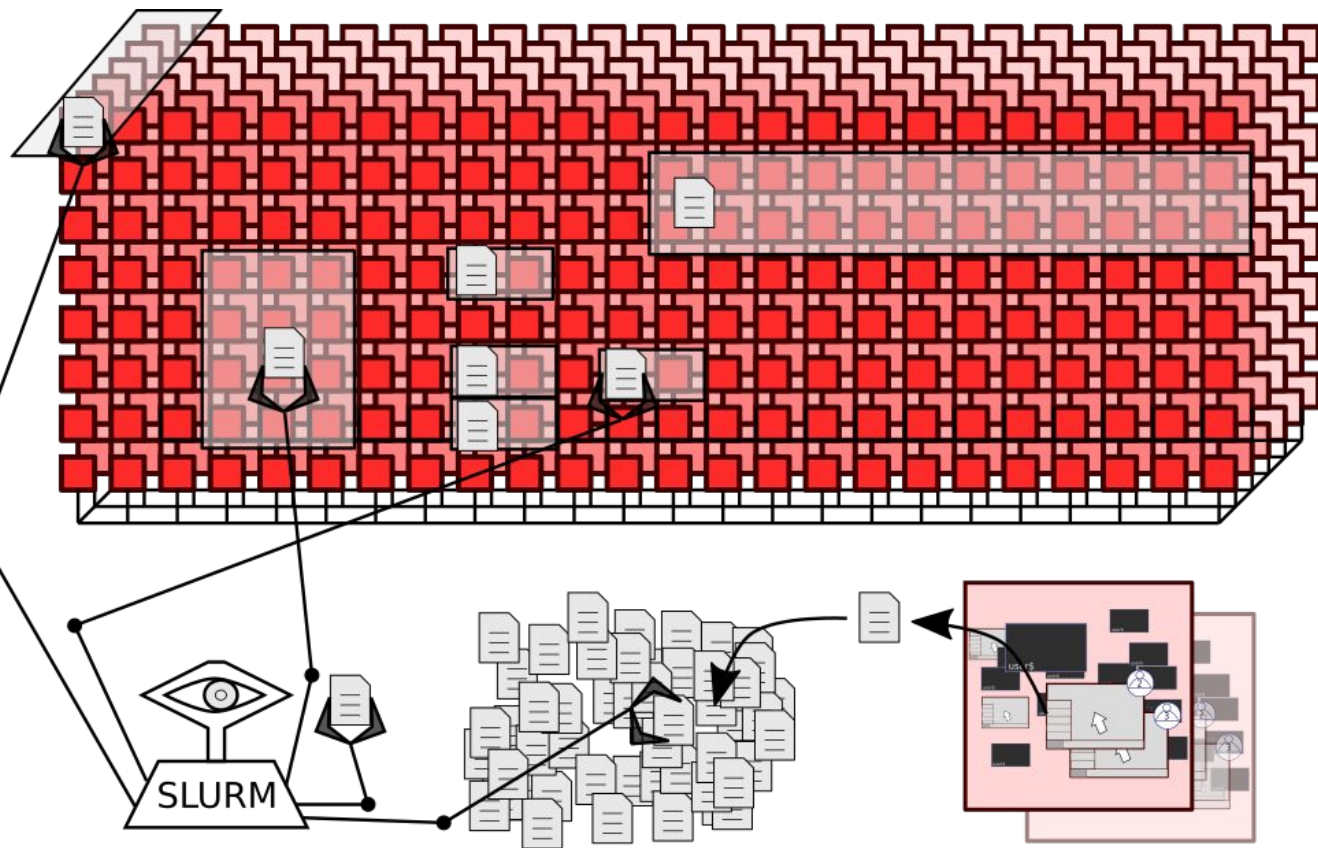
Millions of submitted jobs per year

Many different shapes of jobs  
(single core to thousands, minutes  
to days)

Special requirements



# Queue system: Slurm



- Many jobs
- Many users
- Resource access only through Slurm
- Several methods
  - interactive
  - sbatch
  - (salloc)
- Run as much as possible, order based on prior usage

# Queue system: Slurm Triolith settings

- Fairshare scheduling with backfill (priority to projects with low usage)
- Reservations: `--reservation=devel` (rapid start for short jobs)
- Job length from minutes to 168 hours (7 days)
- Short and wide jobs, aka “flat jobs” should be avoided
- Priority boosting of jobs is available (at a cost)

# Queue system: Running on the compute nodes

- **Interactive** (example: `interactive -n 1 -t 10`)
  - Direct access to nodes, hands-on
  - Good for testing
  - Troubleshooting
- **Batch** (example: `sbatch -t 12:00:00 calculation4b.sh`)
  - Indirect access to nodes
  - Login not needed
  - Only performs job script instructions
  - Output written to files
  - Production type jobs

# Queue system: Submitting jobs

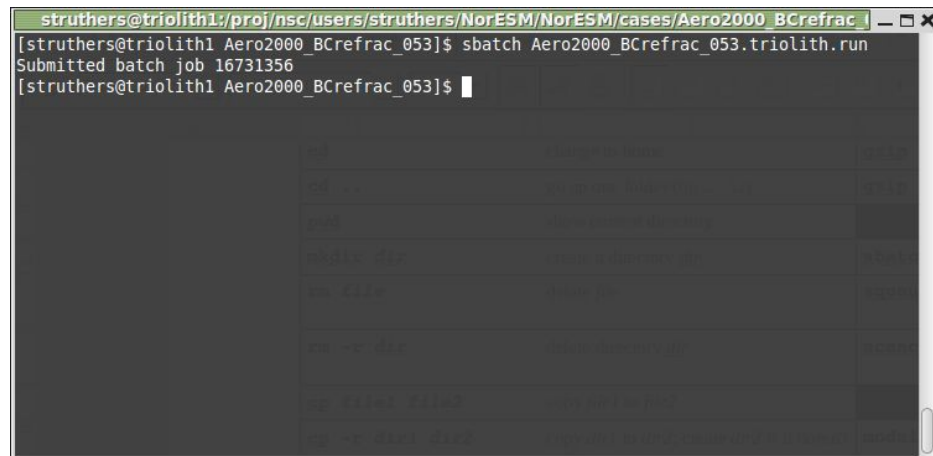
To submit a job to the queue, use the *sbatch* command:

```
sbatch [resource req.] jobfile
```

*job* is a 'batch script' which may contain:

- Slurm resource requests
- model configuration settings
- setting of environment variables
- module commands
- ...

and the actual launching of your application

A terminal window screenshot showing the execution of the `sbatch` command. The terminal title bar reads "struthers@triolith1: /proj/nsc/users/struthers/NorESM/NorESM/cases/Aero2000\_BCrefrac\_053". The prompt is "[struthers@triolith1 Aero2000\_BCrefrac\_053]". The user enters the command `sbatch Aero2000_BCrefrac_053.triolith.run`. The output is "Submitted batch job 16731356". The prompt then changes to "[struthers@triolith1 Aero2000\_BCrefrac\_053]".

```
struthers@triolith1: /proj/nsc/users/struthers/NorESM/NorESM/cases/Aero2000_BCrefrac_053
[struthers@triolith1 Aero2000_BCrefrac_053]$ sbatch Aero2000_BCrefrac_053.triolith.run
Submitted batch job 16731356
[struthers@triolith1 Aero2000_BCrefrac_053]$
```



# Queue system: specifying resources

The minimum information required by the Slurm scheduler:

- How many tasks do you want: `-n` (number of tasks/processes)
- For how long do you want them: `-t hh:mm:ss`

Additional information:

- Which project should be charged for this request if a member of several  
(`-A PROJECTNAME`)
- Jobname (`-J NAME`)
- Request e-mail notifications (`--mail-type=TYPE, BEGIN, FAIL, TIME_LIMIT_90` )

# Queue system: specifying resources

Resources can be requested either on the command line:

```
sbatch -n 128 -t 48:00:00 -A nsc ece-ifs.sh
```

or within the batch script:

```
#SBATCH -n 128  
#SBATCH -t 48:00:00  
#SBATCH -A nsc
```

Note:

- If the model does not complete before the walltime limit (`-t 48:00:00`), the scheduler will terminate the job and valuable data may be lost.

# How to start programs: Many serial programs

There are several ways to start many independent serial instances within one jobscript.

- 1) Use a simple for loop. This quickly becomes complicated and requires some bash coding experience. Tasks started are also limited to the first node (on which your script is executed).
- 2) Use `srun` or Gnu Parallel to start tasks across many nodes

# How to start programs: OpenMP

When running OpenMP programs consider submitting with a matching specification, for example, “-n1 -c8” is one task with eight cores per task

```
$ export OMP_NUM_THREADS=8
```

```
$ ./my_openmp_program ...
```

# How to start programs: MPI

MPI programs cannot be executed as is but require a launcher command to be used. On NSC we provide a launcher called `mpprun` which integrates well (does not require much extra work).

When submitting remember that the `-n` option will map to number of MPI ranks. If you require extra memory per rank use you can use `--ntasks-per-node` (set to less than 16) or `-c` (set to more than 1) or use fat nodes.

```
$ interactive -n 64 -t 10
```

```
...
```

```
$ mpprun ./my_mpi_binary
```

```
mpprun info: Starting impi run on 4 node ( 64 rank X 1 th ) for job ID
```

```
...
```

# Howto start programs: Wrapper provided / special

It is not uncommon for software provided by NSC to come with specific information on how to run it.

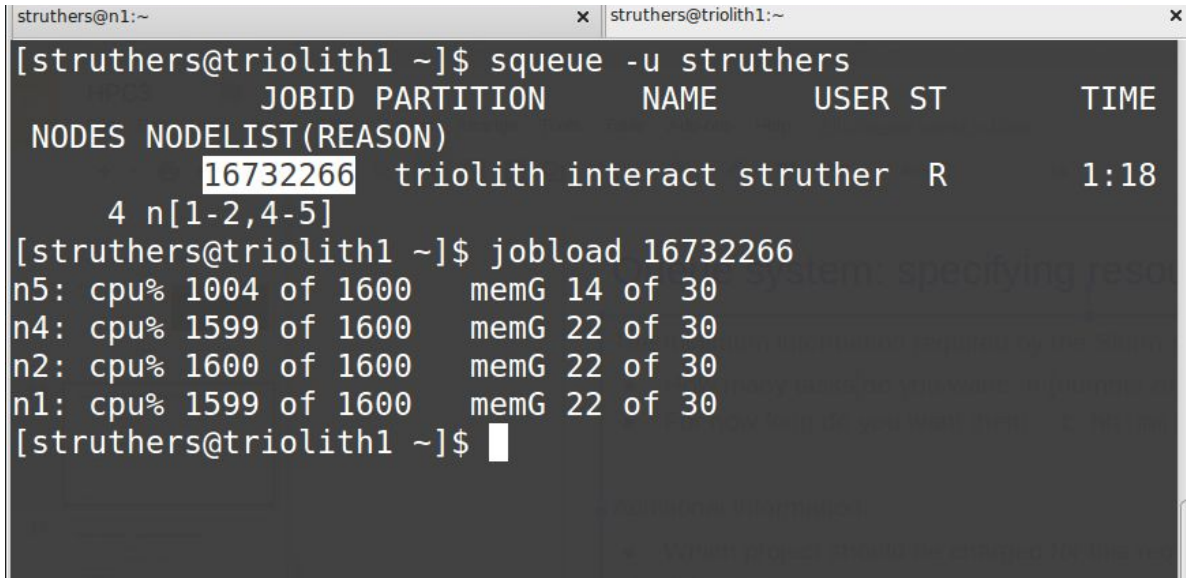
Sometimes it's just a convenient template to start from and other times it documents an application specific startup wrapper.

Examples include Comsol, NWchem, Gaussian

# What's going on with your job?

Some tools which you can use:

- `queue -u username`
- `jobload jobID`
- `lastjobs`



```
struthers@n1:~ x struthers@triolith1:~ x
[struthers@triolith1 ~]$ queue -u struthers
          JOBID PARTITION   NAME     USER ST       TIME
NODES NODELIST(REASON)
          16732266 triolith interact struther R       1:18
           4 n[1-2,4-5]
[struthers@triolith1 ~]$ jobload 16732266
n5: cpu% 1004 of 1600   memG 14 of 30
n4: cpu% 1599 of 1600   memG 22 of 30
n2: cpu% 1600 of 1600   memG 22 of 30
n1: cpu% 1599 of 1600   memG 22 of 30
[struthers@triolith1 ~]$
```

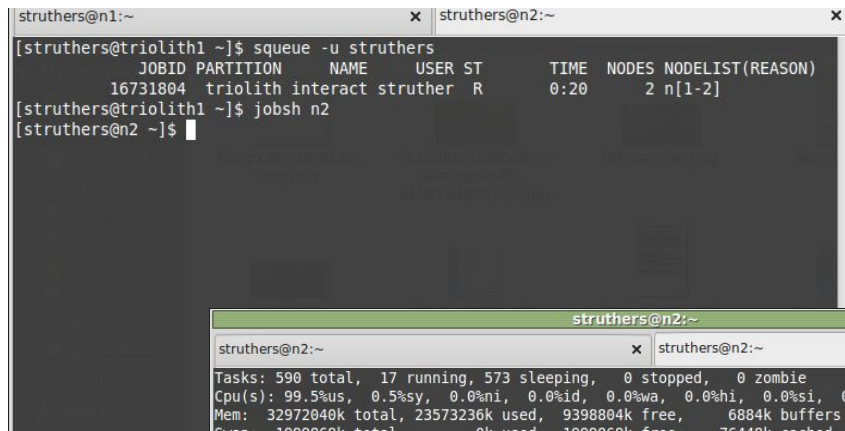
# What's going on with your job?

Login to the job nodes:

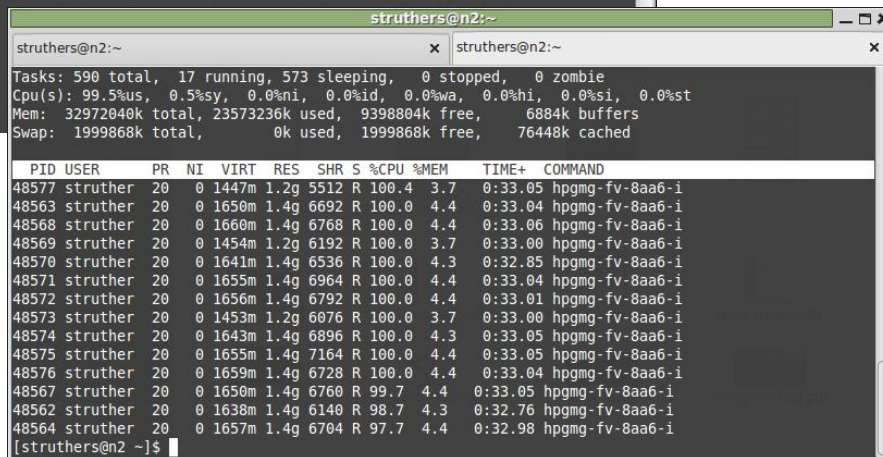
```
queue -u username
```

```
jobsh nodename
```

```
top
```



```
struthers@n1:~ x struthers@n2:~
[struthers@triolith1 ~]$ queue -u struthers
      JOBID PARTITION  NAME  USER ST   TIME  NODES NODELIST(REASON)
      16731804 triolith interact struther R    0:20    2 n[1-2]
[struthers@triolith1 ~]$ jobsh n2
[struthers@n2 ~]$
```



```
struthers@n2:~ x struthers@n2:~
Tasks: 590 total, 17 running, 573 sleeping, 0 stopped, 0 zombie
Cpu(s): 99.5%us, 0.5%sy, 0.0%ni, 0.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 32972040k total, 23573236k used, 9398804k free, 6884k buffers
Swap: 1999868k total, 0k used, 1999868k free, 76448k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
48577 struther  20   0 1447m 1.2g 5512 R 100.4  3.7   0:33.05 hpgmg-fv-8aa6-i
48563 struther  20   0 1650m 1.4g 6692 R 100.0  4.4   0:33.04 hpgmg-fv-8aa6-i
48568 struther  20   0 1660m 1.4g 6768 R 100.0  4.4   0:33.06 hpgmg-fv-8aa6-i
48569 struther  20   0 1454m 1.2g 6192 R 100.0  3.7   0:33.00 hpgmg-fv-8aa6-i
48570 struther  20   0 1641m 1.4g 6536 R 100.0  4.3   0:32.85 hpgmg-fv-8aa6-i
48571 struther  20   0 1655m 1.4g 6964 R 100.0  4.4   0:33.04 hpgmg-fv-8aa6-i
48572 struther  20   0 1656m 1.4g 6792 R 100.0  4.4   0:33.01 hpgmg-fv-8aa6-i
48573 struther  20   0 1453m 1.2g 6076 R 100.0  3.7   0:33.00 hpgmg-fv-8aa6-i
48574 struther  20   0 1643m 1.4g 6896 R 100.0  4.3   0:33.05 hpgmg-fv-8aa6-i
48575 struther  20   0 1655m 1.4g 7164 R 100.0  4.4   0:33.05 hpgmg-fv-8aa6-i
48576 struther  20   0 1659m 1.4g 6728 R 100.0  4.4   0:33.04 hpgmg-fv-8aa6-i
48567 struther  20   0 1650m 1.4g 6760 R 99.7  4.4   0:33.05 hpgmg-fv-8aa6-i
48562 struther  20   0 1638m 1.4g 6140 R 98.7  4.3   0:32.76 hpgmg-fv-8aa6-i
48564 struther  20   0 1657m 1.4g 6704 R 97.7  4.4   0:32.98 hpgmg-fv-8aa6-i
[struthers@n2 ~]$
```



# Queue system

A common question: Why doesn't my job start?

- Requesting the use of a special resource
- Project overuse
- Blocking reservation or scheduled stop
- Incorrect job specification
- Mismatching expectations
- Note: The amount of core-h left in your project does not directly correlate with priority. Other projects can affect where you fall on the priority list.

# Parallel job: Fundamentals

- Amdahl's law

Your parallel performance will always be limited by that part that you cannot parallelize  
(and that part is impossible to completely get rid of)

Parallel part



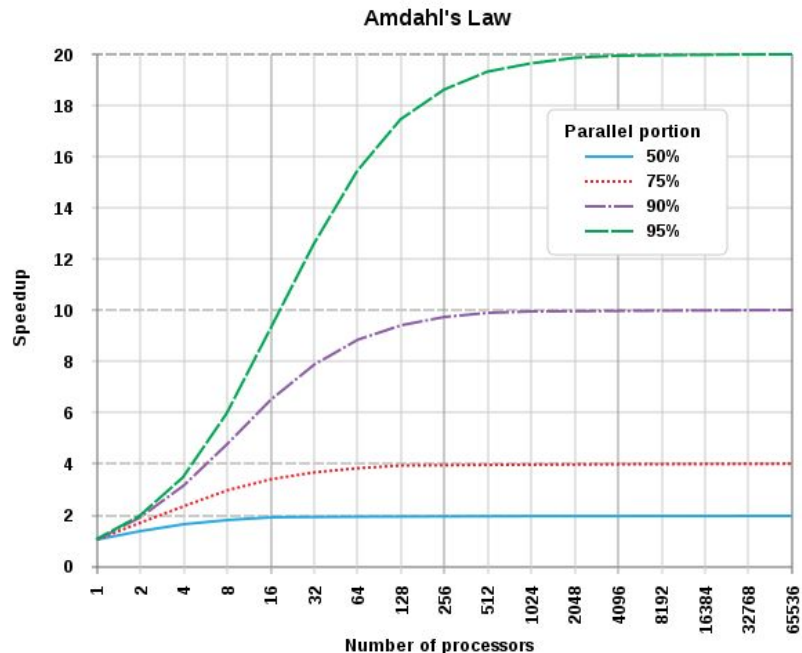
Runtime

$$4 + 24 + 2 = 30$$

$$4 + 12 + 2 = 18$$

$$4 + 6 + 2 = 12$$

$$4 + 2 = 6$$

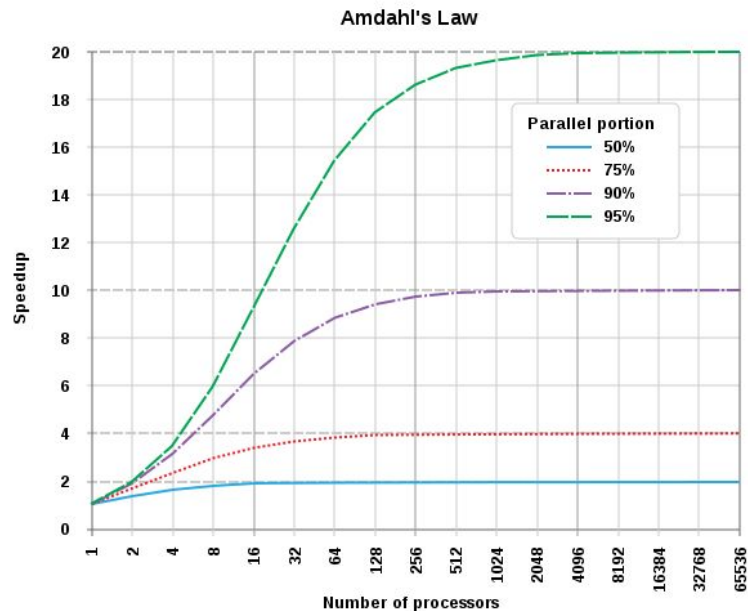


# Parallel job: Fundamentals

The common solution to limited scaling due to Amdahl's law is what is commonly referred to as "Weak scaling"

**Weak scaling** is when you increase your problem size as you increase the amount of resources. It works because the parallelized part typically grows much more than the serial part. Weak scaling is the norm.

The opposite to weak scaling is **strong scaling**. Scaling your application performance while maintaining the same problem size.



# Putting it together: Starting with a new job type

Decide what software to use

- 1) Look for suitable software and a suitable variant and version
- 2) Understand what type of software this is so that a suitable job size can be selected

Also consider how you can efficiently do pre- and post-processing for the jobs. Additional tools/software needed?

# Putting it together: Starting with a new job type

Organize input/output data

- 1) Learn what input and output you will need/get
- 2) Estimate if storage requirements will become challenging (remember both total size and number of files can be problematic)
- 3) Consider what to save, what to archive and what to remove
- 4) Consider where to put work directories for the jobs and name things such that it will be possible to figure out even at a later date

Note: remember that while NSC storage very seldom loses user files it's not perfect and only /home is backup protected.

# Putting it together: Starting with a new job type

Test run the software and:

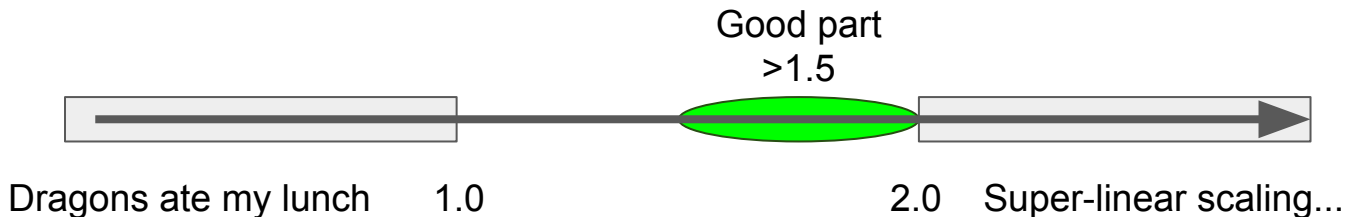
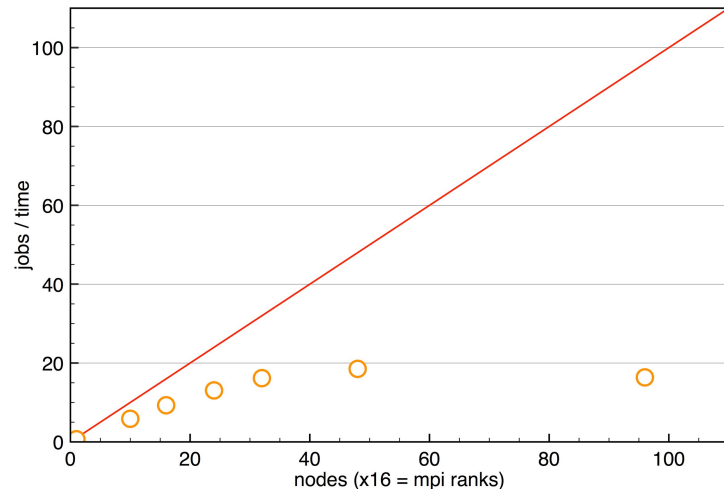
- Verify the output. Does it produce reasonable results?
- Verify that the job launches correctly and seems to run ok (`jobsh`, `jobload`, ...)

You should also perform a simple scaling analysis to make sure that you're using the projects resources efficiently.

# Simple scaling analysis

A minimal scaling analysis can save vast amounts of allocated core hours.

1. Tool your runscript to time your simulation
2. Run an initial best guess number of cores ( $n$ )
3. Run the same test on half the number of cores ( $n/2$ )
4. Score = performance / performance on half the number of cores



# Best practices & suggestions

## General reminders

- Be careful in how you use the Triolith login nodes
- Use SUPR to follow project usage
- Use the NSC documentation
- Don't be afraid to ask for help ([support@nsc.liu.se](mailto:support@nsc.liu.se)) if you need help or have questions



# Best practices & suggestions

## Data/files/storage

- Use shared folders or files in `/proj` for collaborative work
- Use separate working directories for your submitted jobs
- Use sensible names for working directories and files
- Try at least some basic data management:
  - Remove redundant data
  - Compress your data where appropriate
  - Keep down number of files (e.g. use `tar`)
  - Keep track of how much space you use (`snicquota`)


# Best practices & suggestions

## Batch jobs

- Use sensible jobnames
- Save output data (logs) from jobs
- Keep Job IDs - helpful for diagnosing problems
- Try to understand the scaling characteristics of your software before running large jobs

# Best practices & suggestions

## Common problems

- My job didn't run as I expected (or crashed). What do I do now?
  - First, try to understand what the problem might be
  - Write a good support question  **include more details**
- Don't run heavy things / production work on the login node
  - For brief testing e.g. run interactively `--reservation=devel`