Taito User Guide

Autogenerated from HTML pages with pandoc

12/02/19

Contents

Taito User Guide

The super cluster **taito.csc.fi** was taken in use at CSC on 2013. This server is intended for serial and parallel computing tasks that utilize less that 256 computing cores. This guide focuses on the most essential information related to the supercluster taito.csc.fi. New customers of CSC should use this guide together with the *CSC Computing environment user guide* that discusses issues that apply for all the computing servers servers of CSC (e.g. Linux basics and data transport).

Three things you should know before you start using Taito

- Registering as CSC customer allows you to login to Taito, but if you start using Taito **you should join a computing project or open a new computing project.**
- When you log in to address *taito.csc.fi* you end up to one of the login nodes of Taito cluster. These login nodes are **not** intended for heavy computing but for managing your files and batch jobs. Because of that all processes that take more than 1 CPU hour will automatically be killed. Please use **batch jobs** or **taito-shell.csc.fi** for all heavy computing.
- Your home directory has very limited capacity and it should not used for working with large datasets. Please use your **work directory** ($WRKDIR) when you work with large datasets and **HPC-archive** storage system to store and backup the data that you need to preserve.

1.1 Taito supercluster

1.1.1 User policy

The Taito supercluster (taito.csc.fi) is intended for serial and medium sized parallel tasks as well as jobs that require a lot of memory. Researchers that want to use Taito should 1) **register as CSC users** and then 2) **apply for a computing project**. This registration process is described in the chapters 1.2.1 and 1.2.2.2 of the CSC computing environment user guide.

A computing project at CSC has a common computing quota that can be extended by application. Use of Taito or any other server will consume the computing quota granted to the project. **One core hour in Taito consumes 2 billing units** from the computing quota of the project.

The Taito users are allowed to submit up to 1024 simultaneous batch jobs to be executed. The maximum size of a single job is at most 448 compute cores using 672 CPUs.

Table 1.1 Available batch jobs queues in supercluster taito.csc.fi.

Queue	Number of cores	Maximum run time
serial (default)	24 (one node)	3 days
parallel	672 (28 nodes)	3 days
longrun	24 (one node)	14 days
test	32 / 48 (two nodes)	30 min
hugemem	40 (one haswell hugemem node)	7 days/14days

In Taito you don't need to sent the scaling test results of your parallel code to CSC. However, you should still make sure that you are using the resources efficiently i.e. that your code - with the used input - does scale to the selected number of cores. The rule of thumb is that when you double the number of cores, the job should to run at least 1.5 times faster. If it doesn't, you should use less cores. Note that scaling depends on the input (model system) as well as the used code, so you may need to test separately for scaling with the same code for different model systems. If you are unsure, contact CSC Service Desk.

1.1.2 Hardware

Taito (taito.csc.fi) is a 16 cabinet HP cluster based on commodity off-the-shelf building blocks. The theoretical peak performance of the cluster, calculated on the aggregate performance of the computing nodes, is about 600 TFlop/s.

Sandy bridge nodes

As a preparation for the installallation of the new cluster environment, the older part of Taito, consisting of 576 Sandy Bridge nodes, was removed in January 2019.

Haswell nodes

The *Haswell* processors comprise several components: twelve cores with individual L1 and L2 caches, an integrated memory controller, three QPI links, and an L3 cache shared within the socket. The processor supports several instructions sets, most notably the *Advanced Vector Extensions 2* (AVX2) instruction set; however, older instructions sets are still supported. Each Haswell core has dedicated 32KB of L1 cache, and 768 KB of L2 cache. The L3 cache is shared among the processors, and its size is 30 MB. Most of the Haswell nodes (397) have 8 slots of 16GB DDR4 DIMMs, operating at 2133 MHz, for a total of 128GB per compute node. This means that there are 5,3 GB of memory available per core. There are also ten Haswell-equipped nodes with 16 modules of 16 GB 2133 MHz DIMMs, that is 256 GB of DDR4 memory per node, and 10,7 GB per core.

Hugemem nodes

For jobs requiring very large memory, Taito includes six *hugemem* nodes each having **1,5 TB** of memory:

- two HP Proliant DL560 nodes, with 2.7Ghz Sandy Bridge processors with 32 cores (four eight node sockets) and 2 TB of local temporary storage space.
- four Dell R930 nodes, with 2.8Ghz Haswell processors with 40 cores and 2,6 TB of local SSD based fast temporary storage space.

Login nodes

In addition to the computing nodes, Taito has four login nodes, two of which (taito-login3 and taito-login4) are used for logging into the system, submitting jobs, I/O and service usage. The tow other login nodes act as the front ends for the GPGPU and MIC hardware linked to the cluster. The login nodes are HP Proliant DL380 G8.

Interconnect

Communication among nodes and to the storage is done by Infiniband FDR fabric, which provides low latency and high throughput connectivity. High speed interconnect is provided by 58 Mellanox Infiniband FDR switches with 36 ports each, and by the Infiniband HCAs installed on each computing node. The network topology for the cluster is 4:1 pruned tree fabric.

Table 1.2 Configuration of the Taito.csc.fi supercluster. The aggregate performance of the system is 600 TF/s.

Node type	Number of nodes	Node model	Number of cores / node	Total number of cores	Memory / node
Sandy Bridge login node	4	HP Proliant DL380 G8	16	64	64 / 192 GB
Haswell compute node	397	HP Apollo 6000 XL230a G9	24	9528	128 GB
Haswell big memory node	10	HP Apollo 6000 XL230a G9	24	240	256 GB
Sandy Bridge big memory node (these nodes has been removed)	16	HP SL230s G8	16	256	256 GB
Sandy Bridge huge memory node	2	HP Proliant DL560	32	64	1,5 TB
Haswell huge memory node	4	Dell R930	40	160	1,5 TB

The following commands can give some useful information from the whole Taito system or from the current node a user is logged in.

To get a quick overview of all Taito compute node status use the following command:

```
sinfo -Nel
```

The above command prints information in a compute node oriented format. Alternatively, you can get the information in in a partition/queue oriented format with command:

```
sinfo -el
```

For information about the disk systems one can use the following command:

```
df -h
```

Details about the available processors on the current node can be checked with:

```
cat /proc/cpuinfo
```

And details about the current memory usage on the node is shown with:

```
cat /proc/meminfo
```

1.2 Operating system and shell environment

Taito is a Linux cluster. The login nodes are based on the *RedHat Enterprise Linux 6 (RHEL6*) distribution, while the computing nodes use *CentOS 6*, which is a free distribution entirely derived from RHEL6. During the lifetime of the cluster, we aim to keep the software packages up to date following the minor releases of the operating system, as long as this preserves the necessary compatibility with the previous versions. The computing nodes have identical software configuration. The same applies to the login nodes. The system software set installed on the login and computing nodes is relatively minimal but it offers a wide selection of libraries and development packages to compile your own software. In general, all the libraries available on the computing nodes are also available on the login nodes. You can inspect what packages are installed on Redhat based servers using the following command:

```
rpm -qa
```

This command is also useful to find out what is the version of an installed package. Other options can be given to the ***rpm*** command to inspect the system configuration. Alternatively, ***locate*** and ***find*** are also good tools for inspecting the software configuration of a system. Note that users can't use *rpm* command to install software to Taito.

The system packages will be updated during the lifetime of the system without any previous notification. Therefore, we suggest using the module system to load specific library versions and software supported by CSC, or install your own version in *$USERAPPL* directory. In this way your software dependencies will be safely preserved.

As a general rule, x86-64 binaries should be used for software installed on Taito. "x86-64" is the 64-bit extension of the x86 instruction set.

The default and recommended command shell in Taito is **bash**. Previously CSC has been using *tcsh* as the default command shell and you can still continue to use *tcsh* shell in Taito too. If you want to change your default shell in Taito, please contact servicedesk@csc.fi.

When a user logs into Taito, the *bash* start-up script defines a set of CSC specific variables defining the location of the user specific directories: *$WRKDIR, $HOME, $TMPDIR* and *$USERAPPL*. Further, *rm*, *cp* and *mv* commands are aliased so that by default they ask for permission before removing or overwriting and existing files. Also the *clobber options* are set up so that output forwarding does not overwrite an existing file.

If you wish to add more settings or aliases that are automatically set-up when you log in, you should add the corresponding linux commands to the end of the *bash* set-up file *.bashrc* that is located in your home directory.

The Taito system supports UTF-8 character encoding, which makes it possible to represent every character in the Unicode character set. UTF-8 was not supported on older CSC systems, so care should be taken when sharing files with other systems.

Compiling programs should be done in the *$TMPDIR*, which is login node specific instead of *$WRKDIR* or *$HOME* which reside on Lustre file system. Using *$TMPDIR* is much faster for compilation purposes than Lustre and avoids performance degradation for computing jobs.

1.3 Connecting to Taito

To connect Taito, use terminal programs like ***ssh*** (linux, MacOSX) or **PuTTY** (Windows), which provide secure connection to the server. If you are not able to use a suitable terminal program in your local computer

you can use the *SSH console tool* in *Scientist's User Interface*.
For example, when using the ssh command the connection can be opened in the following way:

```
ssh taito.csc.fi -l username
```

If you wish to use applications that use graphics or you want to suspend you session after the day, we recommend that you use the *NoMachine Remote Desktop connection* instead of direct terminal connections.

The Taito supercluster has two *login nodes* (taito-login3.csc.fi and taito-login4.csc.fi). When you open a new terminal connection using the server name *taito.csc.fi* you will end up to one of these login nodes. You can also open the connection directly to two of the login nodes (taito-login3.csc.fi and taito-login4.csc.fi) if needed:

```
ssh taito-login4.csc.fi -l username
```

Note that **login nodes are not intended for heavy computing** but for submitting and managing batch jobs. If you wish to do interactive computing, instead of submitting batch jobs, you can open connection to *taito-shell* (see chapter 8.)

```
ssh taito-shell.csc.fi -l username
```

More details about connecting to the computing severs of CSC can be found from the CSC Computing Environment User Guide chapter 1.3.

1.4 Monitoring the load in Taito

Server information, current CPU load, CPU load history and job count history of Taito supercluster and other CSC's servers can be checked with *Host Monitor* tool of CSC's *Scientist's User Interface* web portal. In addition, signing in offers additional features such as details of all running batch jobs:

- Host Monitor (CSC account required, full access)
- Host Monitor (no authentication, limited functionality)

Once you have logged in to Taito, you can check load status also with commands:

```
squeue
```

and

```
sinfo
```

These commands and tools show the jobs that are currently running or waiting in the batch job system.

1.5 Disk environment

The CSC supercomputing environment allows researchers to analyse and manage large datasets. Supercluster taito.csc.fi and supercomputer sisu.csc.fi have a common disk environment and directory structure where on CSC you can work with datasets that contain several terabytes of data. In Taito (and Sisu) you can store data in several personal disk areas. The disk areas available in Taito are listed in Table 1.2 and Figure 1.2 below. Knowing the basic features of different disk areas is essential if you wish to use the CSC computing and storage services effectively. Note that in Taito all directories use the same Lustre-based file server (except

$TMPDIR which is local to each node). Thus all directories are visible to both the front-end nodes and the computing nodes of Taito.

In addition to the local directories in Taito, users have access to the CSC archive server, *HPC archive*, which is intended for long term data storage. HPC archive server is used through the iRODS software. Projects that have applied cPouta access can also use the CSC Object Storage service that can be be used as common storage area for CSC computing environment, Virtual Machines in cPouta and local computing environment. (See CSC Computing environment user's guide Chapter 3.2 and Chapter 3.3 for more information

Directory or storage area	Intended use	D
$HOME	Initialization scripts, source codes, small data files. Not for running programs or research data.	50
$USERAPPL	Users' own application software.	50
$WRKDIR	Temporary data storage.	5 T
$TMPDIR	Temporary users' files, scratch, compiling.	
project	Common storage for project members. A project can consist of one or more user accounts.	On
HPC archive*	Long term storage.	2,5
Object Storage	Platform independed data stotrage	1 T

*The HPC-archive server is used through iRODS commands, and it is not mounted to Taito as a directory.
** This applies to the files on the login node $TMPDIR. The files in compute node $TMPDIR are kept for the duration of the batch job and deleted immediately after it.

The directories listed in the table above can be accessed by normal linux commands, excluding the archive server, which is used through the *iRODS* software. The $HOME and $WRKDIR directories as well as the HPC archive service can also be accessed through the *MyFiles* tool of the Scientist's User Interface WWW service. The $USERAPPL is a subdirectory of $HOME.

When you are working on command line, you can utilize automatically defined environment variables that contain the directory paths to different disk areas (excluding project disk for which there is no environment variable). So, if you would like to move to your work directory you could do that by writing:

```
cd $WRKDIR
```

Similarly, copying a file *data.txt* to your work directory could be done with command:

```
cp data.txt $WRKDIR/
```

In the following chapters you can find more detailed introductions to the usage and features of different user specific disk areas.

1.5.1 Home directory

When you log in to CSC, your current directory will first be your home directory. Home directory should be used for initialization and configuration files and frequently accessed small programs and files. The size of the home directory is rather limited, by default it is only 50 GB, since this directory is not intended for large datasets.

The files stored in the home directory will be preserved as long as the corresponding user account is valid. This directory is also backed up regularly so that the data can be recovered in the case of disk failures. Taito and Sisu servers share the same home directory. Thus if you modify settings files like *.bashrc*, the modifications will affect both servers.

Inside linux commands, the home directory can be indicated by the *tilde* character (~) or by using the environment variable, *$HOME*. Also the command ***cd*** without any argument will return the user to his/her home directory.

1.5.2 Work directory

The work directory is a place where you can temporarily store large datasets that are actively used. By default, you can have up to 5 terabytes of data in it. This user-specific directory is indicated by the environment variable, $WRKDIR. The Taito and Sisu servers share the same $WRKDIR directory.

The $WRKDIR is NOT intended for long term data storage. Files that have not been used for 90 days will be automatically removed. If you want to keep some data in $WRKDIR for longer time periods you can copy it to directory $WRKDIR/DONOTREMOVE. The files under this sub directory will not be removed by the automatic cleaning process. Please note that the DONOTREMOVE directory is not intended for storing data but to keep available ONLY such important data that is frequently needed. Backup copies are not taken of the contents of the work directory (including DONOTREMOVE directory). Thus, if some files are accidentally removed by the user or lost due to physical breaking of the disk, the data is irreversibly lost.

Please do not use *touch* command particularly if you have lot of files because it is metadata heavy operation and will impact $WRKDIR performance for all users.

$WRKDIR F.A.Q.

- **Q:** Can I check what files the cleaning process is about to remove from my $WRKDIR directory?
  A: You can use command *show_old_wrkdir_files* to check the files that are in danger to be removed. For example the commands below lists the files that are are older than 83 days and thus will be removed after the next seven days.

  ```
  show_old_wrkdir_files 83 > files_to_be_removed
  less files_to_be_removed
  ```

  The first command produces the list and writes it into a file. Please bear in mind that producing the list is a heavy operation so do it only when needed and refer to the file instead.

- **Q:** I've a zip/tar which I've extracted and file dates are old, are those files removed immediately?
  A: No, extracted files will have 90 days grace time.

- **Q:** I've old reference data which I need for verification often. Are those removed?
  A: All files which have been accessed within 90 days are safe (read, open, write, append, etc.). Command: *stat filename* will show timestamps.

- **Q:** How do I preserve an important dataset I have in $WRKIR?
  A: Make a compressed *tar* file of your data and copy it to HPC archive (see chapter 3.2 of the CSC Computing environment user guide).

1.5.3 Software installation directory

Users of CSC servers are free to install their own application software on CSC's computing servers. The software may be developed locally or downloaded from the internet. The main limitation for the software installation is that user must be able to do the installation without using the *root* user account. Further, the software must be installed on user's own private disk areas instead of the common application directories like */usr/bin*.

The *user application directory* ***$USERAPPL*** is a directory intended for installing user's own software. This directory is visible also to the computing nodes of the server, so software installed there can be used in batch jobs. Unlike the work directory, $USERAPPL is regularly backed up.

Sisu and Taito servers have separate $USERAPPL directories. This is reasonable: if you wish to use the same software in both machines you normally you need to compile separate versions of the software in each machine. The $USERAPPL directories reside in your home directory and are called: *appl_sisu* and *appl_taito*. These directories are actually visible to both Sisu and Taito servers. However, in Taito the $USERAPPL variable points to $HOME/appl_taito, and in Sisu to $HOME/appl_sisu.

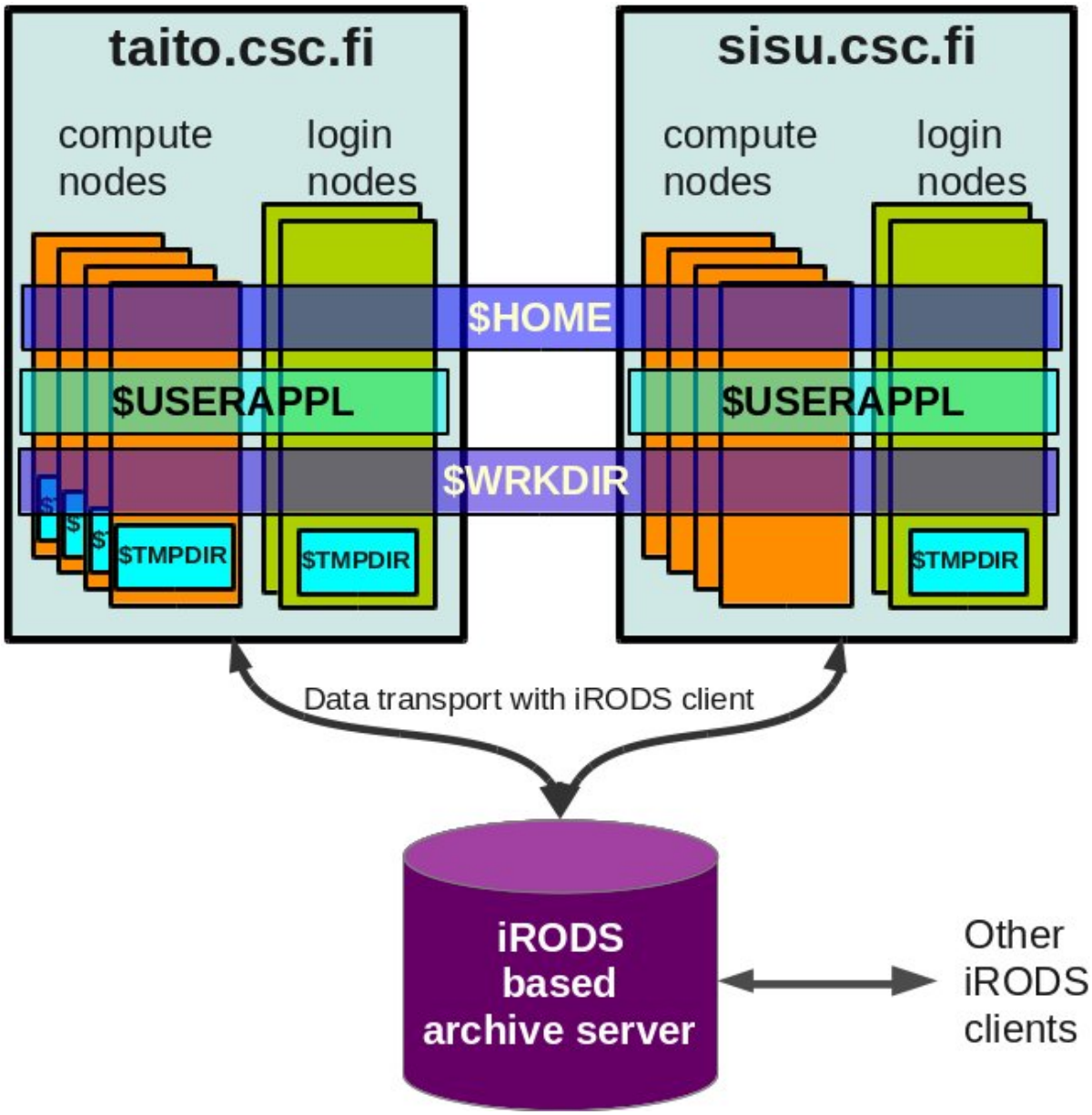

Figure 1.2 Storage environment in Sisu and Taito computers.

1.5.4 Monitoring disk usage

The amount of data that can be stored to different disk areas is limited either by user specific quotas or by the amount of available free disk space. You can check your disk usage and quotas with the command:

```
quota
```

The *quota* command shows also your disk quotas on different areas. If the disk quota is exceeded, you cannot add more data to the directory. In some directories, the quota can be slightly exceeded temporarily, but after a so-called *grace period*, the disk usage must be returned to the accepted level.

When a disk area fills up, you should remove unnecessary files, compress existing files and/or move them to the archive server. If you have well-justified reasons to use more disk space than what your quotas allow, you should send a request to the CSC resource manager (resource_at_csc.fi).

When one of your directories is approaching the quota limit, it is reasonable to check which files or folders take up most space. To list the files in your current directory ordered by size, give command:

```
ls -lSrh
```

Note however, that this command does not tell how much disk space the files in the subdirectories use. Thus it is often more useful to use the command **du** (disk usage) instead. You can, for example, try command:

```
du -sh ./*
```

This command returns the size of each file or the total disk usage of each subdirectory in your current directory. You can also combine *du* with ***sort*** to see what file or directory is the largest item in your current directory:

```
du -s ./* | sort -n
```

Note that as the *du* command checks all the files in your current directory and running the command may in some cases take several minutes.

2. The module system in Taito

Managing the software environment of a cluster with big user base is a complicated task. One have to take into account several, often contradictory, requirements set by different applications, compilers and libraries. For example, it is often necessary to have access to older versions of an application, or to have a library configured with different options for different applications. Setting up a working environment for the user is very difficult when you have several versions of hundreds of applications and libraries. The situation is also difficult for the users looking for a particular application or library.

Environment modules provide a convenient way to dynamically change the user's environment so that different compiler suites and application versions can be used more easily. Modules system modifies the environment variables of the user's shell so that the correct versions of executables are in the path and linker can find the correct version of needed libraries. For example, the command *mpicc* points to different compilers depending of the loaded module. Modules system also provides extensive search commands that can be used to find suitable software modules from the list of installed packages.

Taito uses a recently developed version of environment modules called **Lmod**. It is developed at Texas Advanced Computing Center (TACC) and it is implemented using *Lua* programming language. More technical details can be found from the Lmod homepage. Those who are familiar with the *environment modules* systems used in Sisu should note that the Lmod significantly differs from Sisu's implementation.

2.1 Basic usage

The syntax of the module commands is:

```
module command module-name
```

The currently loaded modules are listed with command:

```
module list
```

For general module information one uses command *module help*. For example, to get more information about loaded module *intel*, one can use command:

```
module help intel
```

New modules can be loaded to your environment using *load* command, for example the *trilinos* module can be loaded using command:

```
module load trilinos
```

Note that you can only load modules that are compatible with other modules that you have loaded. That is, you can not load modules that are conflicting with previously loaded modules, or modules that depend on modules that have not been loaded.

Modules that are not needed or that are conflicting with other modules can be unloaded using *unload* command:

```
module unload mkl
```

Table 2.1 Most commonly used module commands.

Module command	Description
module help *modulename*	Show information about a module.
module load *modulename*	Loads the given environment module.
module unload *modulename*	Unloads the given environment module.
module list	List the loaded modules.
module avail	List modules that are available to be loaded.
module spider *name*	Searches the entire list of possible modules.
module swap *module1 module2*	Replaces a module with a second module.

2.1.1 Finding modules

You can list the modules that are compatible with your current module set by using command:

```
module avail
```

Because of the hierarchical structure of the Lmod system you can not load all installed modules using just one *module load* command. The *avail* command does not show modules that can not be loaded due to conflicts or unmet dependencies. Reason for these protective restrictions is to prevent you from loading module combination that do not work.

You can get the list of all installed software packages using command:

```
module spider
```

You can also give the name or part of the name of the module as an argument, for example:

```
module spider int
```

will list all modules with string “int” in the name. More detailed description of a module can be printed using the full module name with version number, for example:

```
module spider fftw/3.3.2
```

2.1.2 Solving conflicts

As mentioned above, the module system does not let the user to load conflicting modules. In these cases you have to solve these conflicts before loading the module. In the easiest case the module system gives you explicit guidance. For example, if you try to load a compiler module on top of another one, you will get an error message:

```
 -bash-4.1$ module load gcc

Lmod has detected the following error: You can only have one compiler module loaded at a time.
You already have intel loaded.
To correct the situation, please enter the following command:

  module swap intel gcc/4.8.2
```

Some modules depend on other modules. Also in these cases the information from the module system is obvious, for example:

```
-bash-4.1$ module load netcdf4

Lmod has detected the following error: Cannot load module "netcdf4/4.3.0" without these modules loaded
  hdf5-par/1.8.10
```

More complicated procedures are needed when the module is not compatible with currently loaded compiler and/or MPI-library. In these cases the *module avail* command does not even list the module and *module load* command can not find it. Easiest way to check what environment is required for the desired module is to use *module spider* command with version information. For example:

```
-bash-4.1$ module spider hypre/2.9.0b
----------------------------------------------------------------------
 hypre: hypre/2.9.0b
----------------------------------------------------------------------
  This module can only be loaded through the following modules:
       intel/12.1.5, intelmpi/4.0.3
       intel/13.0.1, intelmpi/4.1.0
       intel/13.1.0, intelmpi/4.1.0
...
```

So in this case you will have to load one the listed environments before you can proceed with *module load* command.

2.2 Advanced topics

2.3.1 Module hierarchy

In general, libraries built with one compiler need to be linked with applications using the same compiler. For example, you can not use the MPI Fortran90 module compiled with Intel compilers with *gfortran*, but you have to use a version compiled with *gfortran*. Environment modules have several mechanisms that prevent the user from setting up a non-working environment.

The module hierarchy helps us to keep the compiler and MPI library settings compatible with each other. In practice, for each supported compiler there is a module for a supported MPI library. When user switches the compiler module, the module system tries to find the correct versions of loaded modules:

```
-bash-4.1$ module list
Currently Loaded Modules:
  1) intel/12.1.5 2) mkl/10.3.11 3) intelmpi/4.0.3

-bash-4.1$ module switch intel gcc
Due to MODULEPATH changes the following modules have been reloaded:
 1) mkl/10.3.11 2) intelmpi/4.0.3
```

If a correct version is not found, the module system *deactivates* these modules. In practice, the module is unloaded, but it is marked so that when the compiler/MPI configuration is changed, the system tries to find a correct version automatically.

This hierarchy is implemented by changing the **$MODULEPATH** variable. Every compiler module adds its own path to the module path so that software modules compatible with that specific compiler can be listed. When the compiler module is unloaded, this path is removed from the module path. Same applies also to the MPI modules.

2.2.2 Using your own module files

If you want to use modules to control the software packages that you install by yourself, you can add your own modules files to your home directory. For example, if you add the module files to **$HOME/modulefiles**, you can access them after you add the path to the modules search path using command:

```
module use $HOME/modulefiles
```

3. Batch jobs

CSC uses batch job systems to execute computing tasks in clusters and supercomputers. In this chapter we provide introduction to the **SLURM** (Simple Linux Utility for Resource Management) batch job system that is used in Taito supercluster.

Batch job systems are essential for effective usage of large computing servers. First of all, the batch job system takes care that the server does not get overloaded: Users can submit large amounts of jobs to be executed and the batch job system takes automatically care that optimal number of jobs are running, while rest of the jobs are queueing until sufficient resources are available. Further, most of the batch job systems have a "*fair share*" functionalities that take care that, on the long run, all the users get equal possibilities to

use resources. For example in a case where user *A* has submitted 500 jobs before user *B* submits his job, the user *B* don't have to wait that all the jobs of user *A* have been processed. Instead, the batch job system gives higher priority to the job of user *B* compared to user *A*, as user *A* is already using much more computing resources that user *B*.

When a batch job system is used, the commands to be executed are not started immediately like in normal interactive usage. Instead the user creates a file that contains the Linux commands to be executed. In addition to the commands, this so called *batch job file* normally contains information about the resources that the job needs (for example: required computing time, memory and number of cores). The batch job file is submitted to the batch job system with a *job submission command*. After that the batch job system checks the resource requirements of the job, sends the job to a suitable queue and starts the job when sufficient resources are available. If the job exceeds the requested values (e.g. requires more computing time than what was requested) the batch job system kills the job. After job submission, user can follow the progress of the job or cancel the job if needed.

3.1 Constructing a batch job file

3.1.1 Batch Job Script Wizard

The most common way to use the SLURM batch job system is to first create a *batch job file* that is submitted to the scheduler with command ***sbatch***. You can create batch job files with normal text editors or you can use the *Batch Job Script Wizard* tool, in the *Scientist's User Interface*(https://sui.csc.fi/group/sui/batch-job-script-wizard), (see **Figure 3.1**). In the Batch Job Script Wizard, you first select the server you want to use and then fill in the settings for the batch job. The Batch Job Script Wizard can't directly submit the job, but with the "Save Script" you can save the batch job file directly to your home directory at CSC. After that you can use the *My Files* tool to further edit and launch the batch job (see paragraph 3.1.5).

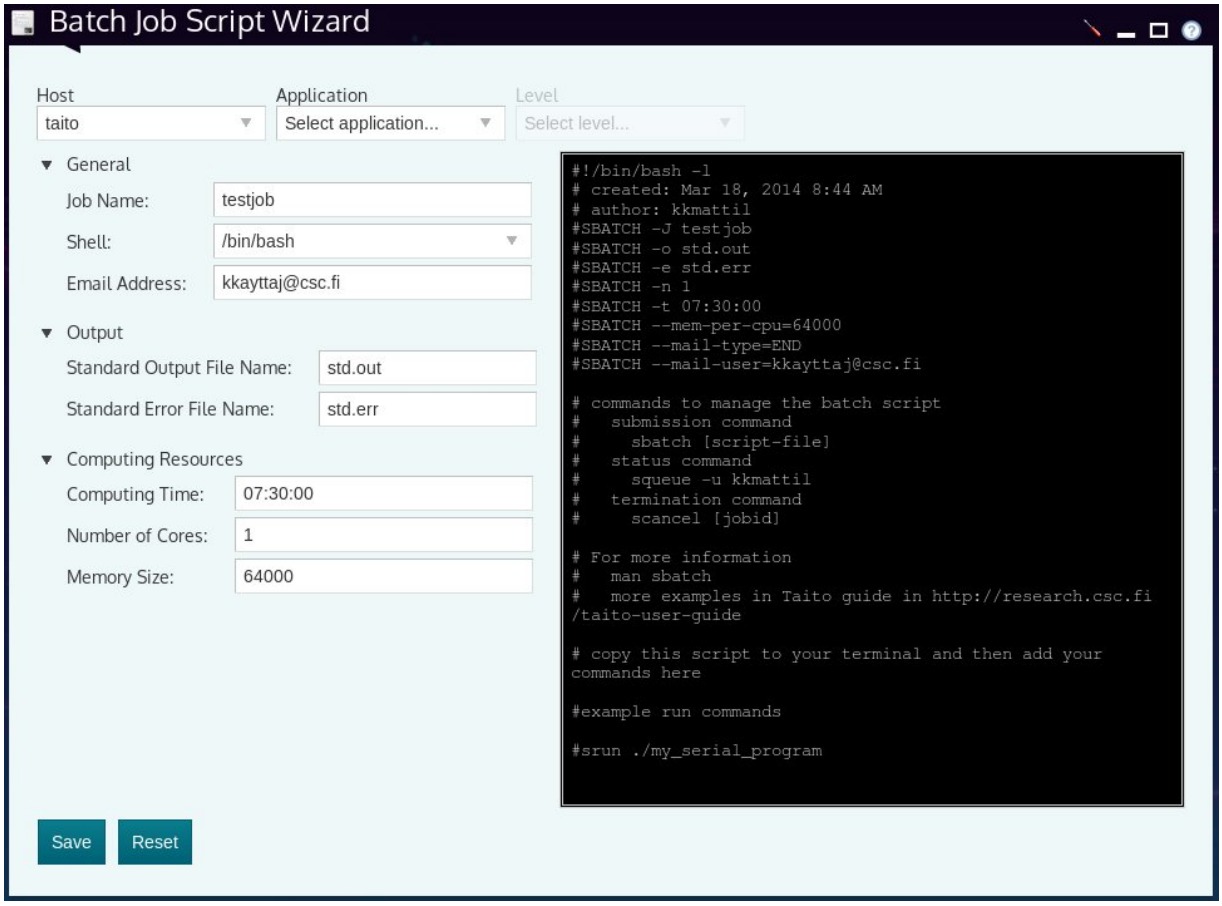

Figure 3.1 Batch Job Script Wizard in the scientist's user interface (https://sui.csc.fi/group/sui/batch-job-script-wizard)

3.1.2 Structure of a batch job file

Below is an example of a SLURM batch job file made with a text editor:

```
#!/bin/bash -l
#SBATCH -J hello_SLURM
#SBATCH -o output.txt
#SBATCH -e errors.txt
#SBATCH -t 01:20:00
#SBATCH -p serial
#
echo "Hello SLURM"
```

The first line of the batch job file (*#!/bin/bash -l*) defines that the *bash* shell will be used. The following five lines contain information for the batch job scheduler. The syntax of the lines is:

```
#SBATCH -sbatch_option argument
```

In the example above we use five *sbatch* options: ***-J*** that defines a name for the batch job (hello_SLURM in this case), ***-o*** defines file name for the standard output and ***-e*** for the standard error. ***-t*** defines that the

maximum duration of the job is in this case 1 hour and 20 minutes. ***-p*** defines that the job is to be send to *serial partition*. After the batch job definitions comes the commands that will be executed. In this case there is just one command: *echo "Hello SLURM"* that prints text "Hello SLURM" to standard output.

The batch job file above can be submitted to the scheduler with command:

```
sbatch file_name.sh
```

The batch job file above includes only the most essential job definitions. However, it is often mandatory or useful to use other **sbatch** options too. The options needed to run parallel jobs are discussed more in detail in the following chapters. Table 3.1 contains some of the most commonly used *sbatch* options. The full list of *sbatch* options can be listed with command:

```
sbatch -h
```

or

```
man sbatch
```

Table 3.1 Most commonly used *sbatch* options

Slurm option	**Description**
–begin=*time*	Defer job until HH:MM MM/DD/YY.
-c, –cpus-per-task=*ncpus*	Number of cpus required per task.
-C, –constraint=*value*	In Taito, the –constraint=hsw option can be used to select hugemem nodes with Haswell pro
-d, –dependency=*type:jobid*	Defer job until condition on jobid is satisfied.
-e, –error=*err*	File for batch script's standard error.
–ntasks-per-node=*n*	Number of tasks to per node.
-J, –job-name=*jobname*	Name of the job.
–mail-type=*type*	Notify on state change: BEGIN, END, FAIL or ALL.
–mail-user=*user*	Who to send email notification for job state changes.
-n, –ntasks=*ntasks*	Number of tasks to run.
-N, –nodes=*N*	Number of nodes on which to run.
-o, –output=*out*	File for batch script's standard output.
-t, –time=*minutes*	Time limit in format *hh*:*mm*:*ss*.
–mem=MB	Maximum amount of real memory per node required by the job in megabytes. (Recommende
–mem-per-cpu=MB	Maximum amount of real memory per allocated CPU required by the job in megabytes.(Rec
-p	Specify queue (partition) to be used. In Taito the available queues are: serial, parallel, longru

In the second batch job example below options *–mail-type* and *–mail-user* are used to make the batch system to send e-mail to address kkayttaj@uni.fi when to job ends. Further the job is defined to reserve 4GB of memory. In the output and error file definitions *%j* is used to use the job id-number in the file name, so that if the same batch job file is used several times, the old output and error files will not get overwritten.

```
#!/bin/bash -l
#SBATCH -J hello_SLURM
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 01:20:00
#SBATCH -n 1
#SBATCH -p serial
#SBATCH --mail-type=END
#SBATCH --mail-user=kkayttaj@uni.fi
#SBATCH --mem=4096
#

echo "Hello SLURM"
./my_command
```

3.1.3 Queues and resource requests

Setting optimal values for the requested computing time, memory and number of cores to be used is not always a simple task. It is often useful to first send short test jobs to get a rough estimate of the computing time and memory requirements of the job. It is safer to reserve more computing time than needed, but remember that jobs with large computing time request may, and often have to, wait longer time in the queue than shorter jobs.

All the batch queues have maximum durations and maximum amount of nodes that a job can use. You can check these limits with command ***sinfo.*** For example:

```
sinfo -o "%10P %.5a %.10l  %.10s %.16F "
PARTITION  AVAIL  TIMELIMIT    JOB_SIZE   NODES(A/I/O/T)
serial*       up 3-00:00:00           1      358/0/0/358
parallel      up 3-00:00:00        1-28      358/0/0/358
longrun       up 14-00:00:0           1      358/0/0/358
test          up      30:00         1-2          2/0/0/2
hugemem       up 7-00:00:00           1          6/0/0/6
```

The *sinfo* output above tells that the cluster has five *partitions* (**parallel**, **serial**, **longrun**, **test** and **hugemem**). For example, the maximum execution time in *parallel* queue is three days (3-00:00:00) and the jobs can use use up to 28 Haswell nodes (28 * 24= 672 cores). Similarly the maximum duration of jobs submitted to *test* queue is 30 minutes (30:00).

The cluster partition you are using should match the reservations for computing time, core number and memory. By default a job is submitted to the *serial* partition, where you can run serial jobs or parallel jobs that use up to 24 cores (one Haswell node) and require at most three days of run time. The maximum memory that can be reserved for a job in the *serial* partition is 256 GB. If your job requests exceeds these limits, you must use option ***-p*** to choose a partition, which meets the resource requests.

For example a serial job that requires 6 days of computing time can be executed in the *longrun* partition

```
#!/bin/bash -l
#SBATCH -J longrun_SLURM
#SBATCH -o output.txt
#SBATCH -e errors.txt
#SBATCH -t 6-00:00:00
#SBATCH -p longrun
#

./my_long_job
```

A small parallel job, that requires 1.0 TB of memory can be executed in the *hugemem* partition

```
#!/bin/bash -l
#SBATCH -J longrun_SLURM
#SBATCH -o output.txt
#SBATCH -e errors.txt
#SBATCH -t 06:00:00
#SBATCH -n 1
#SBATCH --mem-per-cpu=1000000
#SBATCH -p hugemem
#

./my_bigmemory_job
```

Estimating the memory request is even more difficult as it is dependent on several things like algorithm and software and the analysis task. In most case 1-4 GB is enough but you may need to increase the memory size in the case of some application.

Command **sjstat** can be used to check the available memory for nodes in different partitions. The *sjstat* command lists the scheduling pool data and the running jobs. The scheduling pool data can be used to check the available memory in different partitions. You can check just the scheduling pool data by adding option *-c* to the command:

```
sjstat -c

Scheduling pool data:
-------------------------------------------------------------
Pool        Memory  Cpus  Total Usable   Free  Other Traits
-------------------------------------------------------------
serial*    128600Mb    24    348    348      0  hsw,haswell,snb,sandybridge
serial*    258000Mb    24     10     10      0  hsw,haswell,snb,sandybridge
parallel   128600Mb    24    348    348      0  hsw,haswell,snb,sandybridge
parallel   258000Mb    24     10     10      0  hsw,haswell,snb,sandybridge
longrun    128600Mb    24    348    348      0  hsw,haswell,snb,sandybridge
longrun    258000Mb    24     10     10      0  hsw,haswell,snb,sandybridge
test       128600Mb    24      2      2      2  hsw,haswell,snb,sandybridge
hugemem   1551000Mb    32      2      2      0  bigmem,snb,sandybridge
hugemem   1551000Mb    40      4      4      0  bigmem,hsw,haswell,ssd
```

The sample listing above tells e.g. that resource pool *test* contains 2 nodes, each having 128 GB of memory and 24 cores.

Table 3.2 Available batch job queues in supercluster taito.csc.fi.

Queue	Maximum number of cores	Maximum run time
serial (default)	24 (one node)	3 days
parallel	672 (28 nodes)	3 days
longrun	24 (one node)	14 days
hugemem	32/40 (one node*)	7/14 days**
test	48 (two nodes)	30 min

* Sandy Bridge / Haswell (one Sandy Bridge node consists of 32 and Haswell one of 40 cores)
** For exceptionally long hugemem jobs, the maximum run time can be extended up to 14 days using SBATCH option –qos=hugememlong .

3.1.4 Choosing between processor architectures in hugemem queue

If a code is compiled with Haswell processor specific optimization parameters, it will not work in the Sandy Bridge processors. In these cases it is necessary to submit the job so that it will use only Haswell based nodes. This can be specified with by adding following constraint parameter to the batch job file:

```
#SBATCH --constraint=hsw
```

Similarly, if you for some reason want to use only Sandy bridge processors, you should use constraint (Sandy bridge nodes has been removed):

```
#SBATCH --constraint=snb
```

Currently only the **hugemem queue** has nodes with Sandy Bridge processors. By adding defintion –constraint=hsw to your batch job script you can ensure that in the newer Haswell based *hugemem* nodes that have the fast SSD based local temporary storage.

3.1.5 Using Scientist's User Interface to execute batch jobs

My Files tool in Scientist's User interface web portal (https://sui.csc.fi/group/sui/my-files) can be used to transfer and access data in CSC's storage systems (see Chapter 5.1 of CSC computing environment user guide for details). In addition to data management, My Files allows users to submit batch jobs for execution. In My Files, select computing host (for example, Taito) and then browse in $WRKDIR in directory where your job script is saved. Then select job script file and right-click with mouse. This will open a context menu showing action "*Submit Batch Job*". Selecting this action will send your job script for computation.

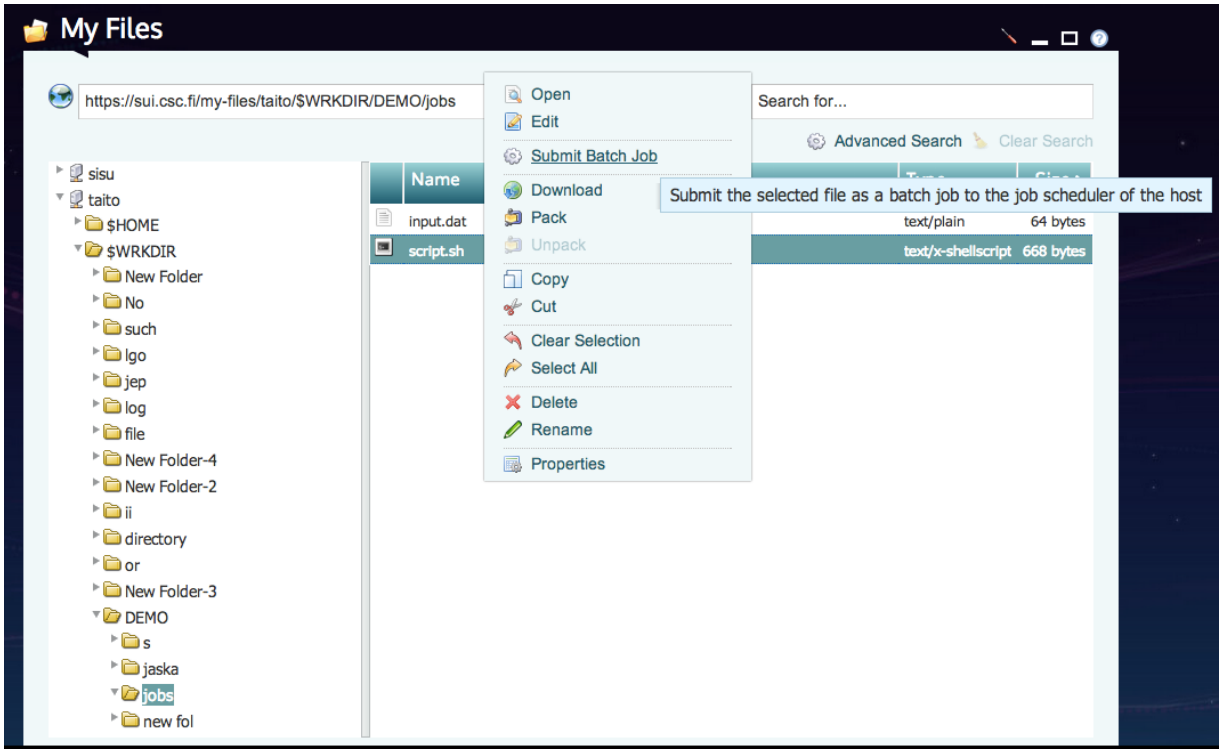

Figure 3.2 Submitting job with My Files in Scientist's User Interface (https://sui.csc.fi/group/sui/my-files)

3.2 Using SLURM commands to execute batch jobs

The basic SLURM commands for submitting batch jobs are **sbatch** that submits jobs to batch job system and **scancel** that can be used to stop and remove a queueing or a running job. The basic syntax of the *sbatch* command is:

```
sbatch -options batch_job_file
```

Normally the *sbatch* options are included in the batch job file, but you can use the options listed in Table 3.1, in command line too. For example:

```
sbatch -J test2 -t 00:05:00 batch_job_file.sh
```

If the same option is used both in command line and in the batch job file, the value defined in the command line overrides the value in the batch job file. When the job is successfully launched, the command prints out a line, telling the *ID number* of the submitted job. For example:

```
Submitted batch job 6594
```

The job ID number can be used to follow the progress of the job or to remove it. For example, a job with ID 6594 can be removed from the batch job system with command:

```
scancel 6594
```

The number of jobs, that a single user can have in the batch job system of Taito at once, has been limited to 896, to prevent batch job system from overloading.

Progress of the submitted batch jobs can be followed with commands *squeue, sjstat* and *sacct*. These commands can also be used to check the status and parameters of the batch job environment. *squeue, sjstat* and *sacct usage* examples are given below.

By default **squeue** command lists all the jobs which are submitted to the SLURM scheduler. If you want to see status of your own jobs only, you can use command:

```
squeue -l -u username
```

or

```
squeue -l -u $USER
```

You can also check the status of a specific job by defining the **jobid** with **-j** switch. Using option **-p *partition*** will display only jobs on that SLURM partition.

Command **scontrol** allows to view SLURM configuration and state. To check when the job waiting in the queue will be executed, the command **scontrol show job *jobid*** can be used. A row *"StartTime=. . . "* gives an estimate on the job start-up time. It may happen that the job execution time can not be approximated, in which case *"StartTime= Unknown"*. Note, that the "StartTime" may change, e.g., be shortened, as the time goes.

The **sacct** command can be used to study the log file of the batch job system. Thus it can show information about both active jobs and jobs that have already finished. By default the *sacct* command shows information about users' own jobs. The *sacct* command has a wide selection of options and parameters that can be used to select the data to be displayed. By default *sacct* displays information from the time period that starts from the midnight of current day. You can change the starting date with option **-S *YYYY-MM-DD***. For example, to list the information since first of February 2015 you can use command:

```
sacct -S 2015-02-01
```

Information about specific jobs can be checked with option **-J** *job-ID*. For example detailed information about job number 6594 could be shown with command:

```
sacct -S 2013-02-01 -j 6594 -l
```

Quite often the full listing of the job information is not desirable. To choose only specific information, you can use option **-o** combined with the list of fields to display. For example:

```
[kkayttaj@taito-login4~]$ sacct -j 6594 -o MaxRSS,AveRSS,ReqMem,Elapsed,AllocCPUS
    MaxRSS     AveRSS     ReqMem    Elapsed AllocCPUS
---------- ---------- ---------- ---------- ---------
                       2347Mc   02:01:49         4
  3480116K   3480116K     2347Mc   02:01:49         1
```

In the example above, the listing shows that job 6594 used 3.5 GB (3480116 KB) of memory and lasted 2 hours, 1 minute and 49 seconds. This information could then be used to optimize batch job parameters for other similar jobs.

When a batch job has finished it is good to run **seff** command to check the efficiency of your job. The syntax of the *seff* command is:

```
  seff jobid
```

A sample session below shows a case where a job (job_id: 54321) took 49 min and 19 s and used the reserved CPU-resources rather efficiently (98.68% efficiency). In the cases of memory, nearly 40 GB was reserved but only bit over 4 GB was used in maximum. Thus for a second similar job, the user should consider decreasing the memory reservation.

```
[kkayttaj@taito-login4~] seff 54321
Job ID: 54321
Cluster: csc
User/Group: kayttaj/somegroup
State: COMPLETED (exit code 0)
Cores: 1
CPU Utilized: 00:48:40
CPU Efficiency: 98.68% of 00:49:19 core-walltime
Memory Utilized: 4.06 GB
Memory Efficiency: 10.39% of 39.06 GB
```

Table 3.1 Most frequently used SLURM commands.

Command	Description
sacct	Displays accounting data for all jobs.
salloc	Allocate resources for interactive use.
sbatch	Submit a job script to a queue.
scancel	Signal jobs or job steps that are under the control of SLURM (cancel jobs or job steps).
scontrol	View SLURM configuration and state.
seff	View the CPU and memory efficiency (real usage compared to the reserved resources)
sinfo	View information about SLURM nodes and partitions.
sjstat	Display statistics of jobs under control of SLURM (combines data from sinfo, squeue and scontrol).
smap	Graphically view information about SLURM jobs, partitions, and set configurations parameters.
squeue	View information about jobs located in the SLURM scheduling queue.
srun	Run a parallel job.

3.3 Parallel batch jobs

Two approaches are commonly used in creating software that are able to utilize several computing cores. Message Passing interface (MPI) based methods and threads based programs (POSIX-threads, OpenMP). Considering CSC resources, Sisu supercomputer is intended for large MPI based parallel jobs but smaller MPI jobs can be run in the Taito supercluster too. In case of threads-based parallel programs, the jobs should be executed mainly in the Taito supercluster.

3.3.1 Threads-based parallel jobs

In case of threads-based parallel computing, the number of parallel processes (threads) is limited by the structure of the hardware: all the processes must be running in the same node. Thus in the *Haswell* nodes Taito cluster, threads-based programs can't use more than 24 computing cores.

Sbatch option –***cpus-per-task***=*number_of_cores* is used the define the number of computing cores that the batch job task will use. Option –**nodes=1** ensures that all the reserved cores will be located in the same node and **-n 1** will assign all the reserved computing cores for the one same task.

In the case of threads-based jobs, the –**mem** option is recommended for memory reservation. This option defines the amount of memory needed per node. Note that if you use –**mem-per-cpu** option instead, the total memory request of the job will be memory request multiplied by the *number-of-cpus*. Thus if you modify the number of cores to be used, you should check the memory reservation too.

Below is a sample batch job that uses bowtie2 software that can use threads-based parallelization.

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH -p serial
#SBATCH --mem=6000
#

module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2 -p $SLURM_CPUS_PER_TASK -x chr_18 -1 y_1.fq -2 y_2.fq > output.sam
```

In the example above, one task (-n 1) that uses 6 cores (–cpus-per-task=6) with total of 6 GB of memory (–mem=6000) is reserved for two hours (-t 02:00:00). All the cores are assigned from one computing node (–nodes=1). When the job starts, the CSC bioinformatics environment, that includes Bowtie2, is first set up with command:

```
module load biokit
```

After that two bowtie2 commands are executed. The indexing command, *bowtie2-build*, does not utilize parallel computing. In case of the *bowtie2* command, the number of cores to be used is defined with option *-p*. In this case we are using six cores so the definition could be: *-p 6*. However in this case we use environment variable $SLURM_CPUS_PER_TASK instead. This variable contains the number of cores defined by the *--cpus-per-task* option. Thus by using $SLURM_CPUS_PER_TASK we don't have to modify the *bowite2-align* command if we change the number of cores to be used with the SBATCH options.

3.3.2 MPI-based parallel jobs

To compile Fortran + MPI code the following command can be used:

```
mpif90 my_mpi_prog.f95 -o my_mpi_program
```

The output executable program *my_mpi_program* is created.

Exemplary script for running MPI-based parallel job:

```
#!/bin/bash -l
###
### parallel job script example
###
## name of your job
#SBATCH -J my_jobname
## system error message output file
#SBATCH -e my_output_err_%j
## system message output file
#SBATCH -o my_output_%j
## a per-process (soft) memory limit
## limit is specified in MB
## example: 1 GB is 1000
#SBATCH --mem-per-cpu=1000
## how long a job takes, wallclock time hh:mm:ss
#SBATCH -t 11:01:00
##the number of processes (number of cores)
#SBATCH -n 24
##parallel queue
#SBATCH -p parallel
## run my MPI executable
srun ./my_mpi_program
```

3.3.3 Interactive MPI-parallel jobs

The output executable program *my_mpi_program* can be run interactively with commands:

```
salloc -n 48 --ntasks-per-node=24 --mem-per-cpu=1000 -t 00:30:00 -p parallel
srun ./my_mpi_program
exit
```

Options:
-n number of processes (number of cores)
–**ntasks-per-node** On Taito there are 24 cores per node.That way your job will be distributed so that the number nodes is minimized
-t running time, wallclock, format **hh:mm:ss** (**hours:minutes:seconds**)
–**mem-per-cpu** per process memory limit (MB)

Other way (one-liner):

```
salloc -n 48 --ntasks-per-node=24 --mem-per-cpu=1000 -t 00:30:00 -p parallel srun ./my_MPI_executable
```

One can also use –**ntasks-per-node** option to control how the job is distributed to the nodes of the cluster.

3.3.4 Choosing between Sandy Bridge or Haswell nodes

Haswell processors can run code optimized for Sandy Bridge processors, but Sandy Bridge processors cannot run Haswell optimized executables. Currently there are two hugemem nodes that still use the the older Sandy Bridge processors. If you have to run binaries that require the older architecture on hugemem partition, you have to add `--constraint=snb` option to the batch job script (Sandy Bridge nodes has been removed).

3.4 Interactive batch jobs

Interactive batch jobs can be used in Taito for running graphical interfaces or other tasks that require input from the user during the execution of the computing task. Note, that you may need to queue for SLURM to allocate these resources, except if you choose *Taito-shell*, see below.

The most easy way to run interactive batch jobs is to use *Taito-shell*, described in chapter 8 of this guide. When a taito-shell session is opened, the session is actually an interactive batch job, that has following properties:

- Unlimited running time
- Computing capacity up to 4 cores per session
- Shared memory of up to 128 GB per session

You can open Taito-shell session, either by directly connecting taito-shell.csc.fi:

```
ssh -X taito-shell.csc.fi -l csc_user_id
```

or if you have already logged in to Taito, you can open a Taito-shell session with command:

```
sinteractive
```

In cases, the you can't use *taito-shell/sinteractive* (e.g. in the case of interactive mpi jobs), you can use the *srun* command to launch interactive batch job sessions. For example command:

```
 srun -n1 -t02:00:00 --x11=first --pty $SHELL
```

would launch interactive batch jobs session, running the default command shell ($SHELL). In the command above, one core (*-n1*) is reserved for two hours (*-t02:00:00*). The definition *–x11=first* sets up the x11 connection so that graphical user interfaces can be used.

Bellow is a sample session where the VMD molecular visualization program is started on an interactive batch job session. Note that after the *srun* command, the commands are not executed in the login node any more. In stead the VMD is now started in one of the computing nodes (node c120 in this case) and thus VMD is not causing extra load to the login node:

```
[kkayttaj@taito-login4 kkayttaj]$ srun -n1 -t02:00:00 --x11=first --pty $SHELL
[kkayttaj@c120 kkayttaj]$ module load vmd
VMD version 1.9.1 is now in use
[kkayttaj@c120 kkayttaj]$ vmd
```

You can also use interactive batch job sessions to test your software. In these cases you can use *srun* as above or alternatively, first the resource allocation with command: *salloc* and the submit jobs with *srun* command.

In the example bellow a request for 48 cores (*-n 48*) with 1 GB/core (*–mem-per-cpu=1000*) for 30 minutes (*-t00:30:00*) is submitted with the *salloc* command. Two full Taito Haswell nodes nodes will be reserved for the interactive job as 24 cores will be used from one node (*–ntasks-per-node=24*). Once the resource allocation is done the Gromacs jobs can be launched using the *srun* command:

```
salloc -n 48 --ntasks-per-node=24 --mem-per-cpu=1000 -t00:30:00 -p parallel
srun mdrun_mpi -s topol1 -dlb yes
srun mdrun_mpi -s topol2 -dlb yes
exit
```

In the example above the *exit* command closes the resource allocation done by the *salloc* command.

3.5 Array jobs in Taito

3.5.1 Defining an array job

In many cases the computational analysis job contains a number of similar independent subtasks. The user may have several datasets that will be analyzed in the same way or same simulation code is executed with a number of different parameters. These kind of tasks are often called as "*embarrassingly parallel*" jobs as the task can be in principle distributed to as many processors as there are subtasks to be run. In Taito this kind of tasks can be effectively run by using the *array job* function of the SLURM batch job system.

In SLURM, an array job is defined by using the option –**array** or -***a*** . For example definition:

```
#SBATCH --array=1-100
```

will launch not just one batch job, but 100 batch jobs where the subjob specific environment variable **$SLURM_ARRAY_TASK_ID** gets values form 1 to 100. This variable can then be utilized in the actual job launching commands so that each subtask gets processed. All the subjobs are launched to the batch jobs system at once and they will be executed using as many processors as there are available.

Note that in Taito the range of the $SLURM_ARRAY_TASK_ID variable is limited between 0 and 1000.

In addition to a defining a job range, you can also provide a list of job index values. For example definition:

```
#SBATCH --array=4,7,22
```

would launch three jobs with $SLURM_ARRAY_TASK_ID values 4, 7 and 22.

You can also add a step size to the job range definition. For example following array job definition:

```
#SBATCH --array=1-100:20
```

would run five jobs with $SLURM_ARRAY_TASK_ID values: 1, 21, 41, 61 and 81.

In some cases it may be reasonable to limit to number of simultaneously running processes. This is done with notation: %*max_number_of_jobs.* For example, in a case were you have 100 jobs but a license for only five simultaneous processes, you could ensure that you will not run out of license with following definition:

```
#SBATCH --array=1-100%5
```

3.5.2 Simple array job example

As a first array job example lets assume that we have 50 datasets (data_1.inp, data_2.inp ... data_50.inp) that we would like to analyze using program *my_prog,* that uses syntax:

```
my_prog inputfile outputfile
```

Each of the subtasks requires less than 2 hours of computing time and less than 4 GB of memory. We can perform all 50 analysis tasks with following batch job script:

```
#!/bin/bash -l
#SBATCH -J array_job
#SBATCH -o array_job_out_%A_%a.txt
#SBATCH -e array_job_err_%A_%a.txt
#SBATCH -t 02:00:00
#SBATCH --mem-per-cpu=4000
#SBATCH --array=1-50
#SBATCH -n 1
#SBATCH -p serial

# move to the directory where the data files locate
cd data_dir
# run the analysis command
my_prog data_"$SLURM_ARRAY_TASK_ID".inp data_"$SLURM_ARRAY_TASK_ID".out
```

In the batch job script the line *#SBATCH –array=1-50* defines that 50 subjobs will be submitted. Other *#SBATCH* lines refer to the individual subjobs. In this case one subjob uses one processor (*-n 1*) max. 4 GB of memory (*–mem-per-cpu=4000*) and can last max. 2 hours (*-t 02:00:00*). However, the total wall clock time needed to process all the 50 tasks is not limited by any sense.

In the job execution commands, the script utilizes *$SLURM_ARRAY_TASK_ID* variable in the definition of input and output files so that the first subjob will run command:

```
my_prog data_1.inp data_1.out
```

second will run command:

```
my_prog data_2.inp data_2.out
```

and so on.

The job can be now launched with command:

```
sbatch job_script.sh
```

Typically not all jobs get into the execution at once. However after a while a large number of jobs may be running in the same time. When the batch job is finished the data_dir directory contains 50 output files.

If you give command **squeue -l** after submitting your array job, you can see that you have one job pending and possibly several jobs running in the batch job system. All these jobs have a *jobid* that contain two parts: *jobid number* of the array job and the *sub-job number*. Directing the output of each subjob into a separate file is recommended as the file system may fail if several dozens of processes try to write into same file at the same time. If the output files need to be merged into one file it can often be easily done after the array job has finished. For example in the case above we could collect the results into one file with command:

```
cat data_*.out > all_data.out
```

In the case of standard output and error files, defined in the SBATCH lines, you can use definitions **%A** and **%a** to give unique names to the output files of each sub-job. In the file names *%A* will be replaced by the *ID of the array job* and *%a* will be replaced by the *$SLURM_ARRAY_TASK_ID*.

3.5.3 Using a file name list in an array job

In the example above, we were able to use $SLURM_ARRAY_TASK_ID to refer to the order numbers in the input files. If this type of approach is not possible a list of files or commands, created before the submission of the batch jobs, can be used. Let's assume that we have a similar task as defined above, but the file names don't contain numbers but are in format data_aa.inp, data_ab.inp, data_ac.inp... and so on. Now we need first to make a list of files to be analyzed, In this case we could collect the file names into file "namelist" with command:

```
ls data_*.inp > namelist
```

In example below we will use command :

```
 sed -n "row_number"p inputfile
```

to read a certain line form the name list file. In this case the actual command script could be following:

```
#!/bin/bash -l
#SBATCH -J array_job
#SBATCH -o array_job_out_%j.txt
#SBATCH -e array_job_err_%j.txt
#SBATCH -t 02:00:00
#SBATCH --mem-per-cpu=4000
#SBATCH --array=1-50
#SBATCH -n 1
#SBATCH -p serial

# move to the directory where the data files locate
cd data_dir
# set input file to be processed
name=$(sed -n "$SLURM_ARRAY_TASK_ID"p namelist)
# run the analysis command
my_prog $name $name.out
```

This example is otherwise similar to the first one, but it will read the name of the file to be analyzed form a file called *namelist*. This value is stored into variable *$name*, which will be used in the job execution command. As the row number to be read is defined by the *$SLURM_ARRAY_TASK_ID,* each data file listed in the file *namelist* will get processed in a different subjob. Note that as we now use the *$name* also in the output definition the output file name will be in format data_aa.inp.out, data_ab.inp.out, data_ac.inp.out... and so on.

3.5.4 Using array jobs in workflows with sbatch_commandlist

In Taito, you can use command **sbatch_commandlist** to execute a list of commands as an array job. This command takes as an input a text file. Each row in this file is executed as an independet sub-task of a array batch job, automatically generated by the *sbatch_commandlist*.

The sytax of this command is:

```
sbatch_commandlist -commands commandlist
```

Options *-t* and *-mem* can be used to modify the time and memory reservation of the subjobs (default 12 h, 8GB).

After submitting an array job, *sabatch_commandlist* monitors the progress of the job and finishes only when the array job has finished. Thus this command can be used in workflows (including batch job scripts), where only certain steps of the workflow can utilize array jobs based parallel computing.

As an example, lets assume we have a gzip compressed tar-archive file *my_data.tgz* containing a directory with a large number of files. To create a new compressed archive, that includes also a md5 checksum file for each file we would need to: (1) un-compress and un-pack *my_data.tgz*, (2) execute *md5sum* for each file and finally (3) pack and compress the *my_data* directory again. The second step of the workflow could be executed using a for-loop, but we could also use the loop just to generate a list of *md5sum* commands, that can be processed with *sbatch_commandist*.

```
#!/bin/bash -l
#SBATCH -J workfow
#SBATCH -o workflow_out_%j.txt
#SBATCH -e workflow_err_%j.txt
#SBATCH -t 12:00:00
#SBATCH --mem=4000
#SBATCH -n 1
#SBATCH -p serial

#open the tgz file
tar zxf my_data.tgz
cd my_data

#generate a list of md5sum commands
for my_file in *
do
  echo "md5sum $my_file > $my_file.md5" >> md5commands.txt
done

#execute the md5commands as an array job
sbatch_commandlist -commands md5commands.txt

#remove the command file and compress the directory
rm -f md5commands.txt
cd ..
tar zcf my_data_with_md5.tgz my_data
rm -rf my_data
```

Note that the batch job script above is not an array job, but it launches a another batch job that is an array job.

Previous chapter	One level up	Next chapter

3.6 Profiling applications using Allinea Performance Reports

Allinea Performance Reports is a performance analytics tool that produces a one-page performance report of your application detailing the amount of time spent in computation, I/O and communication. This information is essential when trying to run the programs using optimal configuration. The detailed user guide can be found from Allinea web pages.

[3.6.1 Using Performance Reports]3.6.1 Using Performance Reports

Set up the environment for analysis by loading the Performance Reports module as follows:

```
module load allinea/reports-6.0
```

The module adds a wrapper to the srun command so that profiling a MPI program can be done by loading the module in the batch job script before the srun command, for example:

```
#!/bin/bash
#SBATCH -ptest
#SBATCH -Chsw
#SBATCH -t5
#SBATCH -n24

module load allinea/reports-6.0

srun ./my_application
```

Results of the profiling is written to a file that is named as *aperf_NNNNN.txt* where *NNNNN* is the slurm job id of your job.

Serial programs can be profiled by adding the command perf-reports –nompi to your command line before the name of your application. Here is an example of serial job script:

```
#!/bin/bash
#SBATCH -ptest
#SBATCH -Chsw
#SBATCH -t5
#SBATCH -n1

module load allinea/reports-6.0

srun perf-report --nompi ./my_application -i input_file
```

In this case the result file will be named as *my_application_1p_2016-04-08_14-10.txt* where the end of the name is a time stamp.

[3.6.2 Examining the results]3.6.2 Examining the results

The result files are quite self explanatory and in the beginning of the report there is a short summary with general suggestions for improving the performance. Allinea web page has some example reports with additional analysis.

Previous chapter	One level up	Next chapter

4. Compiling environment

4.1 Available Compilers

C, C++ and Fortran are the most frequently used programming languages in scientific computing. In Taito supercluster these programming languages can be used through two *compiler suites*. The default compiler package is the *Intel Parallel Studio XE 16.0*. Alternatively you can use *GNU Compiler collection 4.9.3*. To use another compiler package, or another version of a particular compiler package, one should switch the active package using the *module* system (more information about module system in chapter 2).

For example to swap the default Intel compiler to GNU compiler, give command:

```
module swap intel gcc
```

Table 4.1 Compiler suites available in Taito

Compiler suite	Version	Module	Man pages	User Guides
GNU Compiler Collection	4.9.3	gcc	man gcc C/C++ man gfortran Fortra	GCC
Intel Parallel Studio	16.0	intel	man icc C/C++ man ifort Fortran	Intel

To read a man page of a specific compiler, one should execute the *man* command only after having switched to the relevant programming environment.

For Intel Parallel Studio you can find extensive documentation from the Intel Software pages. In the Intel web site you can find the C/C++ documentation under product name*Intel C/C++ User and Reference Guide*and the Fortran77/95 documentation under product name *Intel Fortran User and Reference Guide*.

CSC has created generic commands aliases such as ***f95*** or ***mpif90*** which refer to the compiler commands of the loaded compiler package. For example, in the case of Intel compilers command *f95* refers to Intel F95 compiler, but when the GNU compiler suite is in use the same command refers to *gfortran* command. The aliases are listed in table 4.2.

Table 4.2 Compiler aliases at CSC

Language	compiler command	MPI parallel compiler command
Fortran 77	f77	mpif77
Fortran 95	f95	mpif90
C	cc	mpicc
C++	CC	mpiCC

There are two important factors that should be taken into account when choosing between the compilers: *correctness* and *performance* of the compiled program.

- **Correctness:** Some programs may only produce correct results when compiled with a particular compiler. It is also possible that the program produces wrong results when compiled using aggressive compiler optimizations. It is thus of key importance to always check that the compiled program actually produces correct results.
- **Performance:** One should choose the compiler giving the best performance, while still producing correct results. It is impossible to know ahead of time which compiler is the best for a particular program. One simply has to find the best compiler and its optimal compiler options using a '*generate and check*' method.

Intel and GNU compilers use different compiler options. Detailed list of options for Intel and GNU compiler can be found from man pages when corresponding programming environment is loaded, or in the compiler manuals on the Web (see links above this chapter).

Table 4.3 below lists some good optimization flags for the installed compilers. It is best to start from the safe level and then move up to intermediate or even aggressive, while making sure that the results are correct and that the program has better performance.

Taito has Intel Sandy Brigde and Haswell microarchitecture nodes. To enable full instruction set that these microarchitectures supports, use the option *-xHost* with Intel Compiler and the option *-march=sandybridge* or *-march=haswell* with GNU compiler. Because *-xHost* will generate instructions for the highest instruction set available on the compilation host processor this option wiil generate Sandy Brigde instructions on login nodes. Haswell instructions will be generated when code is compiled on Haswell nodes (and *-xHost* option has been selected *)*. Interactive sessions on compute nodes are explained on Section 3.4 Interactive bacth jobs. Remember that if the compiled code has Haswell instructions such as ***AVX2*** and ***FMA*** it will not run on Sandy Bridge nodes. *Gnu* compilers (*gcc, g++, gfortran*) versions *4.9 and later* and *Intel* compilers (*icc, icpc , ifort*) versios *14 and later* support Haswell specific compiling.

Table 4.3 Simple optimization flags for Intel and GNU compilers.

Optimisation level	Intel	GNU
Safe	-O2 -fp-model precise -fp-model source (Use all three options. One can also use options -fp-model precise -fp-model source with intermediate and aggressive flags to improve the consistency and reproducibility of floating-point results)	-O2
Intermediate	-O2 -xHost (see Remark1)	-O3 -march=native (see Remark1)
Aggressive	-O3 -xHost -opt-prefetch -unroll-aggressive -no-prec-div -fp-model fast=2 (see Remark1)	-O3 -march=native -ffast-math -funroll-loops (see Remark1)
Haswell instructions (binary works only on Haswell nodes) Load also: "module load binutils"	-xCORE-AVX2 -fma	-march=haswell
Haswell and Sandy Bridge instructions (binary works on all nodes)	-xAVX -axCORE-AVX2	Gnu do not support this
Sandy Bridge instructions but tune for Haswell (binary works on all nodes)	Intel do not support this (use above flags)	-march=sandybridge -mtune=haswell

Remark1: If Intel *-xHost* or GNU flag *-march=native* is selected on login/Sandybridge nodes a compiler will generate Sandy Bridge instructions and binary works on all nodes. If Intel *-xHost* or GNU flag *-march=native* is selected on Haswell nodes a compiler will generate Haswell instructions and binary works only on Haswell

nodes.

Do you need more information about Haswell and Sandybridge compiling? See chapter 4.6 Processor architecture specific compiling.

Link time optimization methods are available on Intel and GNU compilers. In GNU case read more fromlink(see option -flto) and in Intel case this link (see IPO optimization).

Table 4.4 Basic options that are common for both Intel and GNU compilers:

Option	Description
-c	Compiles only, produces unlinked object *filename*.o
-o*filename*	Gives the name filename for the executable. Default: **a.out**
-g	Produces symbolic debug information
-I*dirname*	Searches directory *dirname* for for library files specified by -l
-L*dirname*	Searches directory dirname for for library files specified by -L
-l*libname*	Searches the specified library file with the name **lib***libname***.a**
-O*[level]*	Specifies whether to optimize or not and at which level *level*, for example -O0 means turning off optimizations

4.2 Mathematical libraries

4.2.1 MKL (Intel Math Kernel Library)

Intel MKL is a mathematical library collection that is optimized for Intel processors. On Taito, MKL can be used with both Intel and GNU compilers for Fortran and C/C++ programming. However cluster libraries (BLACS, ScaLAPACK and Cluster FFT functions) will work only with *IntelMPI* on Taito.

MKL includes the following groups of routines:

- BLAS (Basic Linear Algebra Subprograms)
- Sparse BLAS
- LAPACK (Linear Algebra PACKage)
- PBLAS (Parallel Basic Linear Algebra Subprograms)
- BLACS (Basic Linear Algebra Communication Subprograms)
- ScaLAPACK (Scalable LAPACK)
- Sparse Solver routines (direct sparse solver PARDISO, direct sparse solver DSS, iterative sparse solvers RCI, preconditioners for iterative solution process)
- Vector Mathematical Functions (VML, arithmetic, power, trigonometric, exponential, hyperbolic, special, and rounding)
- Vector Statistical Functions (VSL, random numbers, convolution and correlation, statistical estimates)
- General Fast Fourier Transform (FFT) Functions
- Cluster FFT functions
- Partial Differential Equations (PDE) support tools (Trigonometric Transform routines, Poisson routines)
- Nonlinear least squares problem solver routines
- Data Fitting functions (spline-based)
- Support Functions (timing, thread control, memory management, error handling, numerical reproducibility)

MKL library has two integer interfaces 32-bit (they call it: LP64) and 64-bit integer (ILP64) interfaces. So if you work with data arrays that have more than 2^{31}-1 elements ILP64 interface is for you.

MKL supports sequential and threaded programming modes. Intel MKL is based on the OpenMP threading. See: Improving performance with threading for more information.

Users Guide and Reference manual

4.2.2 Usage of MKL in Taito

Make sure that *mkl* module has been loaded. If not give command:

```
module load mkl
```

Basic information can be found from link:

- What You Need to Know Before You Begin Using the Intel Math Kernel Library

Load one of the supported compiler environments. Because MKL includes many libraries and programming interfaces a link line can be a long list. To find a correct line for a case, specify your choices using *Intel Math Kernel Library (MKL) Link Line Advisor* tool:

- http://software.intel.com/sites/products/mkl/

Table 4.5 lists the setting you should use with the *Link Line Advisor*.

Table 4.5 *Link Line Advisor* tips for Taito:

Select Intel® product

Intel MKL 11.0 (or another installed version)

Select OS

Linux

Select compiler

Intel or Gnu Fortran or C/C++

Select architecture

Intel 64

Select dynamic or static linking

both options (static/dynamic) are OK

Select interface layer

LP64 or ILP64 (what is the integer range in your code)

Select sequential or multi-threaded layer

sequential and threaded programming modes are supported

Select OpenMP library

This is needed only for threaded applications and/or enable threading in MKL

Select cluster library

Select these If a code is using ScaLAPACK, BLACS or Cluster FFT

Select MPI library

IntelMPI

Select the Fortran 95 interfaces

Select this only if your application has a Intel MKL Fortran module interface. (if your code has a standard BLAS/LAPACK interface you do not select these)

Link with Intel® MKL libraries explicitly

Copy the results (link line and compiler option results) into a *Makefile*. In command line compiling and linking case remove all brackets that the advisor gives (for example if there is a variable (MKLROOT) in brackets then remove the brackets.

When you run an OpenMP (threaded) application there are environment variables for threading control. Before running an application set the OMP_NUM_THREADS variable. For example 16 threads:

```
export OMP_NUM_THREADS=16
```

MKL has also some additional MKL thread control environment variables.

For quick linking Intel compiler supports variants of the *-mkl* compiler option. The compiler links your application using the LP64 interface and it does not use Intel MKL Fortran module 95 interfaces.

- *-mkl* or *-mkl=parallel* to link with threaded MKL
- *-mkl=sequential* to link with sequential MKL
- *-mkl=cluster* to link with cluster libraries that use IntelMPI

for example:

```
f95 -o my_code my_code.f90 -mkl=sequential
```

Add option *-static-intel* if you want to use static linking, dynamic linking is the default.

MKL include files

This link has a include file table (for Fortran and C/C++).

4.3 Using MPI

Message passing interface (MPI) is a flexible parallel programming paradigm, and it is the dominant method of parallelization codes in scientific computing. MPI is suitable for both distributed memory computers and shared memory architectures. In MPI, each task has an address space in memory that other tasks cannot directly access. When exchange of data is needed between tasks, the tasks send '*messages*' to each other.

Intel MPI, MVAPICH2 and OpenMPI are available on Taito, however CSC does not provide support for OpenMPI. The Intel MPI is the default MPI library. The module commands below enable unloading Intel MPI and loading MVAPICH2.

```
module unload intelmpi
module load mvapich2
```

4.3.1 Compiling and linking MPI programs on Taito

There are compiler wrappers **mpif90**, **mpicc** and **mpiCC** for compiling Fortran, C and C++ MPI programs, respectively. These wrappers take care of compiler and linker directives for compiling MPI programs. Users do not need to specify include file locations, MPI libraries or their locations. For example, a MPI program (one source code file) written with the Fortran language ('my_mpi_prog.f95') is compiled and linked as:

```
mpif90 my_mpi_prog.f95 -o my_mpi_prog
```

To compile each of the source code files separately use *-c* flag. The example below will generate the object code files *my_mpi_source1.o* and *my_mpi_source2.o*.

```
mpif90 -c my_mpi_source1.f95
mpif90 -c my_mpi_source2.f95
```

These can be linked to an executable program (my_mpi_prog)

```
mpif90 my_mpi_source1.o my_mpi_source2.o -o my_mpi_prog
```

The **-show** option displays the actual compilation/linking command generated by the wrapper. For example, try:

```
mpif90 -show
```

4.3.2 Include files

In order to use MPI in your program, you need to include the MPI library in your source code by having one of the following lines.

For programs written in Fortran 77 use:

```
include 'mpif.h'
```

and for Fortran 90 or newer use:

```
use mpi
```

For programs written in C/C++ use:

```
#include "mpi.h"
```

4.4 Shared memory and hybrid parallelization

The Haswell compute nodes on Taito contain two twelve core processors (24 cores per node). Hence, it is possible to run shared memory parallel (OpenMP) programs efficiently within a node with twenty four threads at maximum.

4.4.1 How to compile

Both Intel and GNU compilers support OpenMP. Use the following compiler flags enable OpenMP support.

Table 4.6 OpenMP compiler flags

Compiler	Flag
Intel	-openmp
GNU	-fopenmp

Here are examples for OpenMP and mixed (i.e. hybrid) OpenMP/MPI compiling (upper line: Intel compiler, second line: GNU-compiler

```
f95 -openmp -o my_openmp_exe my_openmp.f95
mpif90 -fopenmp -o my_hybrid_exe my_hybrid.f95
```

See OpenMP web site for more information including standards and tutorials.

Include files

For Fortran 77 use following line:

```
include 'omp_lib.h'
```

For Fortran 90 (and later) use:

```
use omp_lib
```

For C/C++ use:

```
#include <omp.h>
```

4.4.2 Running OpenMP programs

The number of OpenMP threads is specified with an environment variable **OMP_NUM_THREADS.** Running a shared memory program typically requires requesting a whole node. Thus, a twenty four thread OpenMP job can be run interactive on Haswell processors as shown in following examples. If you find out that OpenMP sections of your code do not give run-to-run numerical stability try (with Intel compiled code) to set the variable KMP_DETERMINISTIC_REDUCTION=yes.

Sample session for Intel compiled OpenMP program:

```
export KMP_AFFINITY=compact
export KMP_DETERMINISTIC_REDUCTION=yes   #(if necessary and intel compiler version is 13 or later)
export OMP_NUM_THREADS=24
salloc -N1 --cpus-per-task=24 --mem-per-cpu=1000 -t 01:00:00
srun ./my_openmp_exe
exit
```

Sample session for GNU compiled OpenMP program:

```
export OMP_PROC_BIND=TRUE
export OMP_NUM_THREADS=24
salloc -n 1 -N 1 --cpus-per-task=24 --mem-per-cpu=1000 -t 01:00:00
srun ./my_openmp_exe
exit
```

The corresponding batch queue script would be (below just for Intel compiled OpenMP program):

```
#!/bin/bash -l
#SBATCH -J my_openmp
#SBATCH -e my_output_err_%j
#SBATCH -o my_output_%j
#SBATCH --mem-per-cpu=1000
#SBATCH -t 01:00:00
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --cpus-per-task=24
export KMP_AFFINITY=compact
export KMP_DETERMINISTIC_REDUCTION=yes    #(if necessary and intel compiler version is 13 or later)
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASKS
srun ./my_openmp_exe
```

In the above example replace "export KMP_AFFINITY=compact" with "export OMP_PROC_BIND=TRUE" if a code is compiled by GNU compiler. The KMP_DETERMINISTIC_REDUCTION do not help with GNU compiled code.

4.4.3 Hybrid parallelization

In many cases it is beneficial to combine MPI and OpenMP parallelization. More precisely, the inter-node communication is handled with MPI and for communication within the nodes OpenMP is used. For example, on Haswell, consider an eight-node job in which there is one MPI task per node and each MPI task has twenty four OpenMP threads, resulting in a total core (and thread) count of 192. Running a hybrid job can be done interactively as above with the exception that more nodes are specified and for each node one MPI task is requested. The **parallel** partition must be requested to run the program because there are more than one node. That is, for a 8 x 24 job the following flags are used

```
export KMP_AFFINITY=compact
export KMP_DETERMINISTIC_REDUCTION=yes   #(if necessary and intel compiler version is 13 or later)
export OMP_NUM_THREADS=24
salloc -p parallel -N 8 -n 8 --cpus-per-task=24 --mem-per-cpu=1000 -t 02:00:00
srun ./my_hybrid_exe
exit
```

The corresponding batch queue script would be (for Intel compiled code):

```
#!/bin/bash -l
#SBATCH -J my_hybrid
#SBATCH -e my_output_err_%j
#SBATCH -o my_output_%j
#SBATCH --mem-per-cpu=1000
#SBATCH -t 02:00:00
#SBATCH -N 8
#SBATCH -n 8
#SBATCH --cpus-per-task=24
#SBATCH -p parallel

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASKS

export KMP_AFFINITY=compact
export KMP_DETERMINISTIC_REDUCTION=yes  #(if necessary and intel compiler version is 13 or later)
srun ./my_hybrid_exe
```

In the above example replace “export KMP_AFFINITY=compact” with “export OMP_PROC_BIND=TRUE” if a code is compiled by GNU compiler. The KMP_DETERMINISTIC_REDUCTION do not help with GNU compiled code.

4.4.4 Binding threads to cores

The compilers on Taito support thread/core affinity which binds threads to cores for better performance. This is enabled with compiler-specific environment variables as follows (Sandy Bridge):

Intel:

```
export KMP_AFFINITY=compact
```

GNU:

```
export OMP_PROC_BIND=TRUE
```

If one does not set these variables and values all threads in a node might run in a same core. Learn more about Intel thread affinity interface. Support information for GNU compilers for tread affinity can be found here.

4.5 Debugging Parallel Applications

4.5.1 TotalView debugger

TotalView is a debugger with graphical user interface (GUI) for debugging parallel applications. With TotalView you can:

- run an application under TotalView control
- attach to a running application
- examine a core file

4.5.2 Debugging an Application

Set up debugger environment

```
module load totalview
```

Compile the application to be debugged, for example Fortran, c or C++ program. The compiler option *-g* is generating the debug information.

```
mpif90 -g -o myprog mycode.f90
mpicc -g -o myprog mycode.c
mpiCC -g -o myprog mycode.C
```

Enter a launching command to start TotalView on the application you want to debug. For example 8 core half an hour session in parallel partition

```
salloc -n 8 -t 00:30:00 -p parallel totalview srun -a ./my_parallel_prog
```

Totalview Startup Parameters window may appear. Just clik *Ok* button (in a basic case). TotalView Root and Process window appear. Click the GO button in the Totalview process window. A pop-up window appears, asking if you want to stop the job

Process srun is a parallel job. Do you want to stop the job now

Select *Yes* in this pop-up window.

4.5.3 Very basic features of Totalview

The Process Window contains the code for the process or thread that you're debugging. This window is divided into panes of information. The Stack Trace Pane shows the call stack of routines. The Stack Frame Pane displays all of a routine's parameters, its local variables, and the registers for the selected stack frame.

The left margin of the Source Pane displays line numbers. An ARROW over the line number shows the current location of the program counter (PC) in the selected stack frame. One can place a breakpoint (left mouse click) at any line whose line number is contained within a box. After setting one or many breakpoints *Go* button executes your code to the next breakpoint. When one is placing a breakpoint on a line, TotalView places an icon over the line number. To remove a breakpoint just click the breakpoint icon one more time.

To examine or change a value of a variable right click the variable and select *Dive* from the pop up menu. To see the values of that variable on all processes select *Across Processes* from the pop up menu. A new window will show the values and other information from that variable. On that window one can edit the variable values.

Stepping commands *Go*, *Next*, *Step*, *Out* and *Run to* (on the top of Process window) are controlling the way one is executing the code. *Go* means go to the next breakpoint, if a breakpoint locates inside a loop the next breakpoint is the same until loop ends. *Next* executes the current line and the program counter (arrow) goes to next line. *Step* executes one line in your program and if the source line or instruction contains a subroutine or function call, TotalView steps into it. *Out* executes all statements within subroutine or function and exits. *Run To* executes all lines until the program counter reaches the selected line. A line is selected by clicking a code line (not the line number) and the background of that line turn grey.

4.5.4 Debugging running application

It is also possible to "attach" TotalView to an application which is already running. The job can be interactive or a batch job.

First find out the nodes where your application is running. That can be done if the identication number of the job is known. The command squeue displays information about your SLURM jobs.

```
squeue -u $USER
```

With job-id-number option *squeue* displays information about a chosen job.

```
squeue -j <job id> -l
```

The variable "NODELIST" will contain the nodes where the job is running, for example if NODELIST=n[92-93], the nodes are n92 and n93.

Start TotalView (remember to run setup command *module load totalview* before launching TotalView).

```
totalview
```

From the New Program Window (see image, red boundary line) select "*Attach to process*", then from the "*On host*" line select "*Add host*" (green boundary line) and enter the first node name from from the list

"NODELIST". The processes from the selected node will appear. Select (double-click) the **upper srun** process (blue boundary line). From the Root Window select (double click) rank 0 process (red boundary line). From the Stack Trace Pane (green boundary line) of the Process Window one can select a source code of a routine. You are ready to see/search for the location of the program counter (arrow) and to set breakpoints.

4.5.5 Documents

Latest documents: Totalview documentation | Roque Wave

On a Totalview user interface click Help -> Documentation.

4.6 Processor architecture specific compiling

After the hardware extension in January 2015, Taito has two types of compute nodes:

- **Intel Sandybridge**: 2 Cpus per node, all together 16 cores (original Taito, has been removed)
- **Intel Haswell**: 2 Cpus per node, all together 24 cores (extension part)

Haswell procesors have some new instructions such as ***AVX2*** and ***FMA***, that are not available in the Sandy Bridge processors. This means that it is possible to compile a binary in such a way that it won't run on *Sandybridge* nodes but might run very well on *Haswell* nodes. To get best performance on new and old nodes a code may need two binaries, one for *Sandybrigde* and another for *Haswell*. The binaries that are optimized just for one architecture are below called single code path binaries. But of course Sandybridge optimized code can be run on Haswell nodes too.

Gnu compilers (*gcc, g++, gfortran*) versions *4.9 and later* and *Intel* compilers (*icc, icpc , ifort*) versions *14 and later* support Haswell specific compiling. Compilers may support multiple code paths. That means that a binary can have a *baseline path* and one or more additional *optimized paths* i.e. compiler creates both Sandybrigde and Haswell paths. Below are some examples for single and multiple tuned code path options for Intel and GNU compilers.

Please note, that you should use usual optimization switches like *-O2*, *-O3*, or *-funroll-all-loops* too, even though these options are not used in the examples described below.

4.6.1 Intel compiler environment

Recent intel compilers of Taito , for example intel/15.0.0, support single and multiple code paths. *Baseline code path* is determined by the architecture specified option ***-x***. Multiple, *feature specific code paths* are are created with option ***-ax***. These options are available with *icc*, *icpc* and *ifort* compilers and also with MPI wrappers (*mpicc*,*mpiCC*, *mpif90*) when intel environment (later than 14) is loaded. Compiler will generate a single code path that support AVX2 and FMA instructions if the options ***-xCORE-AVX2*** and/or ***-fma*** are chosen. An example, this binary will run only on Haswell nodes.

```
icc -xCORE-AVX2 -fma -o example example.c
```

Multiple code path example below has a baseline path switch ***-xAVX*** (for Sandybridge) and two other feature specific paths ***-axCORE-AVX2,CORE-AVX-I*** (for Haswell and IvyBridge) Option *-ax* tells the compiler to generate one or more versions of functions that will utilize features that these instructions have. This binary will run on Sandybridge (baseline code path) and Haswell nodes (feature specific code path).

```
 icc -xAVX -axCORE-AVX2,CORE-AVX-I -o example example.c
```

4.6.2 GNU compiler environment

GNU compilers, for example gcc/4.9.1, do not support multiple code paths but compiler can tune the generated code for specified cpu-type. Option ***-march*** will choose the generated instructions (for example Sandybridge instructions) but with switch ***-mtune*** compiler can schedule things so that it runs faster on Haswell cpu. These options are available *gcc*, *g++* and *gfortran* compilers and and also with MPI wrappers (*mpicc*, *mpiCC*, *mpif90*) when GNU environment (later than 4.9) is loaded.

Single code path example that will run only on Haswell nodes.

```
 gcc -march=haswell -o example example.c
```

Tuned code path example that will run both on Sandybridge and Haswell nodes.

```
gcc -march=sandybridge -mtune=haswell -o example example.c
```

As always if good performance binary is the target it is worthwhile to try how above mentioned switches might work with your code.

4.6.3 Login nodes

Login nodes have Sandybridge architecture. Remember this if you apply compiler options like *-xHost* (Intel compiler) or *-march=native* (GNU Compiler). If compilation is done on a login node the binary will have Sandybridge single code path and if compilation is done on a Haswell node the binary will have a Haswell single code path.

4.7 Profiling Applications with Intel Tools

4.7.1 Serial and Multithreaded Applications

Intel VTune Amplifier is a powerful profiling tool that can be used to collect performance data of your application. It is best suited to be used with serial and multithreaded code.

Using Intel VTune Amplifier

Set up the environment for profiling by loading the VTune module as follows:

```
module load intel-vtune/16.1
```

If you want to get source code level information, compile your code with optimizations enabled and add also the debugging information option *-g*. Basic hotspot analysis is the first analysis type you should try. Here is a sample batch job script that can be used to profile a serial and OpenMP applications:

```
#!/bin/bash
#SBATCH -ptest
#SBATCH -Chsw
#SBATCH -t5
#SBATCH -n1
#SBATCH -c4

module load intel-vtune/16.1

srun amplxe-cl -r results_dir_name -collect hotspots -- ./my_application
```

Analyzing Results Using GUI

Results can be viewed using the **amplxe-gui** application. Unfortunately it does not work well with *ssh* and X11 forwarding, so we recommend using the analysis tool in *NoMachine* environment (see NoMachine user's guide). The GUI is available in Taito when the module *intel-vtune/16.1* is loaded. You can inspect the results of a profile run by giving the name of the results directory as an argument to the amplxe-gui, for example, the results of previous example can be viewed with command amplxe-gui results_dir_name. Please see Intel's documentation for more information on using the GUI: https://software.intel.com/en-us/amplifier_2015_help_lin

4.7.2 MPI Applications

Intel MPI library provides a simple and light-weight profiling for applications. The collected results include the number of different communication calls, amount of transferred data, etc. The statistics collection can be enabled using an environment variable I_MPI_STATS, for example:

```
export I_MPI_STATS=1
```

Different integer values control the output level, for example value 1 gives results for amount of transferred data, value 2 adds to the results also the number of different communication calls and so on. When the statistics collection is turned on, the results for each task are written to an output file stats.txt. For more details, see the MPI library documentation at Intel's site: https://software.intel.com/en-us/node/528838

Intel Thread Analyzer and Collector

For more detailed analysis you can use *Intel Thread Analyzer and Collector* (ITAC), which can be used to visualize the MPI communication and identify the hotspots and scaling bottle-necks. In order to use the thread analyze make sure that you have module *intel-TraceCollector* loaded using command module load intel-TraceCollector. Also make sure that you are using Intel MPI, that is, you have some version of module intelmpi loaded. Compile you program code using the mpi wrapper compilers (mpicc, mpiCC and mpif90) and add options -g -trace to the linking options. For example:

```
mpicc -g -trace -O3 -xCORE-AVX2 my_program.c
```

After compilation you can run your program code as you would normally do. Note that the tracing can produce huge log files, so it is preferable to use as small and short test case as possible. It is also possible to use the ITAC programming API to collect data only for a portion of the code. See the Intel documentation for more instructions: https://software.intel.com/en-us/articles/intel-trace-analyzer-and-collector-documentation

Analyzing the Trace Results

After the collecting is done, a series of log files with .stf. suffix will be created in the working directory. They can be viewed and analyzed using the Trace Analyzed GUI called *traceanalyzer*, which is available on the Taito login nodes when *itac* module is loaded. You can pass the name of the trace file as an argument, for example traceanalyzer my_program.stf, or open the results file using the GUI. You should also consider using NoMachine (see the VTune instructions for more information). Please see the documentation at Intel's site for detailed usage instructions of Trace Analyzer: https://software.intel.com/en-us/articles/intel-trace-analyzer-and-collector-documentation

5. User's own software installations

The users of CSC servers are free to install their own application software to Taito. The software to be installed may be developed locally or downloaded from the internet. The main limitation for the software installation is that the user must be able to do the installation without using *root* user account or *sudo* command. Further, the software must be installed to users own private disk areas instead of the common application directories like */appl/bin* or */usr/bin*. If root access is needed, e.g. in the case of repository installations, you an consider using the Pouta could computing service in stead of Taito.

In Taito, *user application directory* **$USERAPPL** is a directory that is intended for installing users own software tools. Taito and Sisu severs have separate $USERAPPL directories. Like the work directory, this directory is visible to the computing nodes of the server too so software installed there can be used in batch jobs. Unlike the work directory $USERAPPL is permanent and backuped directory. Thus you do not need worry about preserving you software installation. It will stay available for you, until you remove the executables.

Below are three examples of using the $USERAPPL directory.

Example 1. Installing your own version of MCL program

Example 2. Installing and using your own Perl module in Taito

Example 3. Installing and using your own Python module in Taito

6. Using Taito-GPU

This part of the guide gives a short introduction to compiling, running, profiling and debugging applications that can utilize NVIDIA Tesla K80 and P100 GPUs installed on Taito.

GPGPU stands for *General-Purpose computation on Graphics Processing Units*. GPUs can be used to accelerate computationally intensive code. This is done together with CPUs by offloading some compute-intensive parts of the application to the GPUs. GPGPU implementation is well suited for data parallel and throughput intensive parts of the application. Data parallelism in this case means that GPU can execute the selected operation on different data elements simultaneously. GPUs have thousands of cores for this task (NVIDIA Tesla P100 has 3584 cuda cores). Also it is necessary to have lots of data otherwise the job is not a throughput intensive.

6.1 Taito-GPU hardware and operating system

The Taito cluster includes a separate partition of compute nodes with dedicated GPU accelerator cards. Currently in the system there are 26 nodes containing 4 Pascal P100 GPUs each and 12 nodes with 2 dual GPU K80 GPUs each.

The P100 nodes consists of 26 Dell PowerEdge C4130 servers with:

- 2x Xeon E5-2680 v4 CPUs with 14 cores each running at 2.4GHz
- 512 GB of DDR4 memory
- 4x P100 GPUs connected in pairs to each CPU
- 2x800GB of Sata SSD scratch space

The K80 nodes consists of 12 Dell PowerEdge C4130 servers with:

- 2x Xeon E5-2680 v3 CPUs with 12 cores each running at 2.5GHz
- 256 GB of DDR4 memory
- 2x K80 GPU cards each with 2 GPUs for a total of 4 GPUs per node, these are all connected to the first CPU
- 850GB of HDD scratch space

The different kinds of nodes are connected to each other using FDR InfiniBand which is then connected to the Taito fabric allowing the use of the same storage environment. Note that the P100 nodes can be moved between Taito and the Pouta cloud environment meaning not all 26nodes will be present all the time in Taito. The nodes share the same operating system and offer a similar modules environment as the Taito cluster, with the some of the modules in Taito-GPU being built and optimized for GPU usage.

6.2 Getting access to Taito-GPU

In order to access Taito-GPU you need to login to **taito-gpu.csc.fi** system first. Only that system currently provides you access to the appropriate module environments (e.g. CUDA-environment and CUDA-aware MPI libraries, and PGI-compiler for OpenACC) to build & launch your GPU application. Please see further sections for how to select the correct modules, how to compile your application and how to run it.

6.3 Module and storage environment on Taito-GPU

Taito-GPU shares the same module and storage environment as the rest of the Taito cluster system. The actual module tree is, however, optimized for GPU usage and thus different. Regarding file system directories, see more from Chapter 1.5 of Taito User Guide.

Please pay attention to your module environment. Due to dynamic linking nature of application program it is recommended that you always have exactly the same environment when running your application as when you used to compile & link it.

For cuda programs use the `cuda-env` module, this module will load cuda, a compatible version of the GCC compiler and a compatible MPI implementation. An example of how to load the cuda-env module:

```
module purge
module load cuda-env
```

If you opt for directive based OpenACC approach, use the `openacc-env` module to load a working enviroment:

```
module purge
module load openacc-env
```

In case of multi-GPU MPI application it is advisable to enable also CUDA-aware message passing. This allows the MPI implementation to directly process device pointers with no need by the user to transfer them from device to host before the transfer. A version of OpenMPI that is built with GPU support is automatically loaded with the `cuda-env` and `openacc-env` modules.

6.4 Compiling & linking GPU-programs

6.4.1 Introduction

In order to make application runnable on GPUs one has to use CUDA programming language, the OpenCL programming language or the OpenACC directive based approach.

For more information on these approaches look at:

CUDA

https://github.com/csc-training/CUDA/blob/master/course-material/intro-to-cuda-csc.pdf

https://devblogs.nvidia.com/parallelforall/even-easier-introduction-cuda/

OpenACC

https://devblogs.nvidia.com/parallelforall/getting-started-openacc/

6.4.2 CUDA

The CUDA compiler is called using the

```
nvcc
```

command. The CUDA compiler will take care of the device code compilation and pass the rest on to the C/C++/Fortran compiler loaded which can be changed by loading different versions of the gcc module.

When compiling CUDA code one should pass the compiler what compute capability the target device supports. In the Taito cluster currently there are three types of different GPUs that all support different compute capabilities. The K80 GPUs support compute capability 3.7 and the newest P100 GPUs support compute capability 6.0. To tell the compiler what compute capability to target for compute capability 3.7 (K80) use:

```
-gencode arch=compute_37,code=sm_37
```

and for compute capability 6.0 (P100) use:

```
-gencode arch=compute_60,code=sm_60
```

The gencode argument can be repeated multiple times so to compile for all architectures present in the system use:

```
-gencode arch=compute_37,code=sm_37 -gencode arch=compute_60,code=sm_60
```

6.4.3 CUDA and MPI

Building CUDA programs with MPI can be done in two ways, one can either point the nvcc compiler to use the mpic++ compiler wrappers instead of gcc directly as a backend compiler or manually include and link MPI using the nvcc compiler.

To change what backend compiler nvcc uses pass the `-ccbin` flag to nvcc:

```
nvcc -ccbin=mpic++ cuda-mpi.cu
```

Alternatively one can do the linking normally done by the `mpic++` wrapper by hand and not need to change the backend compiler:

```
nvcc -I$MPI_ROOT/include -L$MPI_ROOT/lib -lmpi cuda-mpi.cu
```

Please note that the actual include paths and libraries depend on the MPI library.

6.4.4 OpenACC

6.4.4.1 C/C++ OpenACC

To compile and link the application, the PGI environment must be used. Unless you have strict requirement for using C++ compiler (`pgc++`), it is often more convenient to deploy PGI's C-compiler with c99 standard extensions:

```
module purge

module load pgi/16.1 cuda/7.5
module list

pgcc -c99 -O3 -Minfo=all -acc -ta=tesla:cc35,7.5,kepler+ daxpy.c -o daxpy.x.acc
```

Here we only target compute capability 3.5, since PGI does not have special support for 3.7. The code will still also work on K80 cards. Since we asked for compiler info via `-Minfo=all` , we will be rewarded by a rather exhaustive listing of messages.

6.4.4.2 Fortran OpenACC

Compilation goes as follows:

```
module purge
module load pgi/16.1 cuda/7.5
module list

pgfortran -O3 -Minfo=all -acc -ta=tesla:cc35,7.5,kepler+ daxpy.F90 -o daxpy.x.acc
```

6.5 Running GPU-programs

6.5.1 Introduction

It is always recommended to use SLURM batch job file for running GPU specific jobs. However, for quick tests also `srun` command would be acceptable. When running GPU applications the batch queue (or partition) in concern need to be either "*gpu*“, ”*gputest*“, ”*gpulong*" Which partition used is set using

```
-p queue_name
```

What type of GPU used is set using a generic resource (GRES) flag:

```
--gres=gpu:type:n
```

Where type is the type of GPU requested per compute node, currently valid values are k80 or p100 and n is the number of GPUs to reserve per node. On the K80 and P100 nodes one can reserve up to 4 GPUs. The gpu and gpulong partition is intended for production runs while gputest is intended for short (less than 15 min) test and development runs.

To request 2 K80 GPUs use `--gres=gpu:k80:2` or to request 4 P100 GPUs use `--gres=gpu:p100:4`

6.5.2 Running under GNU environment on one GPU

Here is a valid script for running under GNU environment that will default any GPU available:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -p gpu
#SBATCH -t 00:05:00
#SBATCH -J gpu_job
#SBATCH -o gpu_job.out.%j
#SBATCH -e gpu_job.err.%j
#SBATCH --gres=gpu:k80:1
#SBATCH

module purge
module load gcc cuda
module list

srun ./your_binary
```

6.5.3 Running under PGI environment on one GPU

The batch job script is very similar as in the GPU environment, the only difference is that different module needs to be loaded. And executable name is also different.

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -p gpu
#SBATCH -t 00:05:00
#SBATCH -J gpu_job
#SBATCH -o gpu_job.out.%j
#SBATCH -e gpu_job.err.%j
#SBATCH --gres=gpu:k80:1
#SBATCH

module purge
module load pgi cuda/7.5
module list

srun ./your_binary
```

6.5.4 Running under GNU environment on multiple GPUs

Here is a valid script for running under GNU environment, on 2 GPUs on a K80 node:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -p gpu
#SBATCH -t 00:05:00
#SBATCH -J gpu_job
#SBATCH -o gpu_job.out.%j
#SBATCH -e gpu_job.err.%j
#SBATCH --gres=gpu:k80:2
#SBATCH

module purge
module load gcc cuda
module list

srun ./your_binary
```

Here is a valid script for running under GNU environment, on 4 GPUs on K80 a node:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -p gpu
#SBATCH -t 00:05:00
#SBATCH -J gpu_job
#SBATCH -o gpu_job.out.%j
#SBATCH -e gpu_job.err.%j
#SBATCH --gres=gpu:k80:4
#SBATCH

module purge
module load gcc cuda
module list

srun ./your_binary
```

6.5.5 Using the SSD scratch space

Each P100 node is equipped with 800GB of fast SSD storage (Sata SSDs) intended as a scratch space. While the K80 nodes have 850 GB of space consisting of a regular hard drive. This space is placed under the *${TMPDIR}* directory and in that directory there will be a folder with the name of the slurm job id, accessible from the *${SLURM_JOB_ID}* enviroment variable. It is this folder that should be used as a scratch space so to access it easily use *${TMPDIR}/${SLURM_JOB_ID}*. The space is local to each compute node meaning that it is shared among all the users of that node and in multi node jobs each node only has access to its own local scratch space and this cannot be accessed from other compute nodes.
To use this space the user needs to move data there in the beginning of the job script and any data the user wants to retain needs to be transferred from the scratch space before the job finishes, once the job ends the files on the SSD space are removed.

If you are moving a lot of smaller files to the scratch space it the recommended work flow is:

- You first make a tar file out of them, outside of any batch job, place this tar file in the *${WRKDIR}* directory.
- Then as part if your batch job you copy the tar file to the scratch space.

- And finally still as part if your batch job you extract that tar file still to the scratch sapce.

For single node jobs data can be transferred to the *${TMPDIR}* location with normal file copy commands. These commands need to be run in the job script as you do not have access to the *${TMPDIR}* directory from outside the job. Below is an example for a single node script.

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -p gpu
#SBATCH -t 00:05:00
#SBATCH -J gpu_job
#SBATCH -o gpu_job.out.%j
#SBATCH -e gpu_job.err.%j
#SBATCH --gres=gpu:p100:1
#SBATCH

module purge
module load cuda-env/10
module list

cp ${WRKDIR}/your_file.csv ${TMPDIR}/${SLURM_JOB_ID}/your_file.csv
srun ./your_binary
```

For jobs using multiple compute nodes the copy command needs to be run on all nodes participating in the job, so we need to use srun to run the commands.

```
#!/bin/bash
#SBATCH -N 2
#SBATCH -n 2
#SBATCH -p gpu
#SBATCH -t 00:05:00
#SBATCH -J gpu_job
#SBATCH -o gpu_job.out.%j
#SBATCH -e gpu_job.err.%j
#SBATCH --gres=gpu:p100:1
#SBATCH

module purge
module load cuda-env/10
module list

srun -N ${SLURM_JOB_NUM_NODES}-${SLURM_JOB_NUM_NODES} -n ${SLURM_JOB_NUM_NODES} cp ${WRKDIR}/your_file

srun ./your_binary
```

6.6 Deploying GPU + MPI-programs

It is possible to utilize multiple GPUs to the benefit of better application performance. Normally this is done by creating an MPI-program and making sure each MPI-task is connected to a GPU.

When we have Hyper-Q/MPS activated on Taito-GPU system, each GPU can be shared between multiple MPI-tasks of the same host CPU.

The MPI message passing is usually done by sending and receiving messages between host CPUs. However, with the advent of new CUDA-aware MPI libraries (like OpenMPI & MVAPICH2) it is possible to exchange messages between GPU resident data, so being able to pass the MPI implementation a pointer to memory resident on the GPU.

This may be beneficial, since not only your code becomes more readable (e.g. OpenACC versions), but also you avoid all the hassle in updating back & forth the data on host CPU just for the sake of making the message passing to work.

When using multiple nodes one needs to be extra verbose in the batch script to ensure the correct placement of the tasks. The easiest way to ensure correct placement is to use the `--ntasks-per-node` flag to specify how many task each node should be running. The `--gres` flag should be set to how many GPUs per node the job will use.

For example to run 4 tasks on 2 nodes and use 2 GPUs per node the following batch script should be used:

```
#!/bin/bash
#SBATCH -N 2
#SBATCH -n 4
#SBATCH -p gpu
#SBATCH -t 00:05:00
#SBATCH -J gpu_job
#SBATCH -o gpu_job.out.%j
#SBATCH -e gpu_job.err.%j
#SBATCH --gres=gpu:k80:2
#SBATCH --ntasks-per-node= 2
#SBATCH

module purge
module load gcc cuda
module list

srun ./your_binary
```

6.7 Profiling GPU-programs

6.7.1 Introduction

In order to monitor performance of your GPU application, the best bet is to use NVIDIA's "nvprof" command line tool.
It shows you how much time is spent in CUDA-kernels and data transfers between host CPU and GPU. It does not show performance of the host part of your application.

6.7.2 Using nvprof

The nvprof is unbelievably easy to use. You don't have to recompile nor relink your application. Only put "nvprof" in front of your application invokation and you are done. Profiling output will be written in a human readable form immediately after completion of your application.

The CUDA-only application gives the following output :

```
srun -N1 -pgpu --gres=gpu:k40:1 nvprof daxpy.x.gnu
==117693== NVPROF is profiling process 117693, command: daxpy.x.gnu
n=134217728 : vlen=256 : griddim = 524288 1 1 : blockdim.x = 256 1 1
daxpy(n=134217728): sum=9.0072e+15 : check_sum=9.0072e+15 : diff=0
==117693== Profiling application: daxpy.x.gnu
==117693== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 78.52%  1.35003s         2  675.01ms  668.75ms  681.27ms  [CUDA memcpy HtoD]
 20.49%  352.28ms         1  352.28ms  352.28ms  352.28ms  [CUDA memcpy DtoH]
  0.99%  16.967ms         1  16.967ms  16.967ms  16.967ms  daxpy(int, double, double const *, double*)

==117693== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 87.15%  1.72141s         3  573.80ms  369.60ms  682.45ms  cudaMemcpy
 12.73%  251.40ms         2  125.70ms  1.2711ms  250.12ms  cudaMalloc
  0.09%  1.8257ms         2  912.83us  839.03us  986.63us  cudaFree
  0.02%  311.92us        83  3.7580us     269ns  126.07us  cuDeviceGetAttribute
  0.00%  68.179us         1  68.179us  68.179us  68.179us  cudaLaunch
  0.00%  47.282us         1  47.282us  47.282us  47.282us  cuDeviceTotalMem
  0.00%  39.486us         1  39.486us  39.486us  39.486us  cuDeviceGetName
  0.00%  13.732us         4  3.4330us     237ns  11.644us  cudaSetupArgument
  0.00%  3.2010us         1  3.2010us  3.2010us  3.2010us  cudaConfigureCall
  0.00%  2.8320us         2  1.4160us     505ns  2.3270us  cuDeviceGetCount
  0.00%  1.1070us         2     553ns     505ns     602ns  cuDeviceGet
```

Whereas running OpenACC C-version of DAXPY against “nvprof” gives :

```
% srun -N1 -pgpu --gres=gpu:k40:1 nvprof daxpy.x.acc
==115197== NVPROF is profiling process 115197, command: daxpy.x.acc
daxpy(n=134217728): sum=9.0072e+15 : check_sum=9.0072e+15 : diff=0
==115197== Profiling application: daxpy.x.acc
==115197== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 50.68%  18.245ms         1  18.245ms  18.245ms  18.245ms  daxpy_27_gpu
 31.15%  11.213ms         1  11.213ms  11.213ms  11.213ms  init_13_gpu
 17.81%  6.4099ms         1  6.4099ms  6.4099ms  6.4099ms  sum_up_35_gpu
  0.35%  127.00us         1  127.00us  127.00us  127.00us  sum_up_36_gpu_red
  0.01%  3.0080us         1  3.0080us  3.0080us  3.0080us  [CUDA memcpy DtoH]
  0.00%  1.5990us         1  1.5990us  1.5990us  1.5990us  [CUDA memcpy HtoD]

==115197== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 86.57%  272.72ms         1  272.72ms  272.72ms  272.72ms  cuCtxCreate
 11.44%  36.034ms         8  4.5042ms  1.7240us  18.252ms  cuStreamSynchronize
  1.35%  4.2681ms         5  853.63us  147.42us  2.0846ms  cuMemAlloc
  0.36%  1.1258ms         1  1.1258ms  1.1258ms  1.1258ms  cuMemAllocHost
  0.18%  571.83us         1  571.83us  571.83us  571.83us  cuModuleLoadData
  0.06%  193.58us         4  48.394us  20.160us  125.87us  cuLaunchKernel
  0.02%  52.670us         1  52.670us  52.670us  52.670us  cuStreamCreate
  0.01%  29.421us         1  29.421us  29.421us  29.421us  cuMemcpyDtoHAsync
  0.01%  19.862us         1  19.862us  19.862us  19.862us  cuMemcpyHtoDAsync
  0.00%  13.031us        14     930ns     373ns  2.5170us  cuPointerGetAttribute
  0.00%  2.7230us         4     680ns     330ns  1.6490us  cuModuleGetFunction
  0.00%  2.6310us         2  1.3150us     431ns  2.2000us  cuDeviceGetCount
  0.00%  2.0510us         1  2.0510us  2.0510us  2.0510us  cuCtxAttach
  0.00%  1.8740us         1  1.8740us  1.8740us  1.8740us  cuCtxSetCurrent
  0.00%  1.1190us         2     559ns     555ns     564ns  cuDeviceGet
  0.00%  1.0990us         2     549ns     297ns     802ns  cuCtxGetCurrent
  0.00%     815ns         1     815ns     815ns     815ns  cuDeviceComputeCapability
```

7. Using Taito-shell for running interactive jobs in Taito

7.1 What is Taito-shell

The *Taito-shell* environment is intended for **interactive use of scientific applications maintained by CSC**, **serial jobs** and usage of **graphical application interfaces**. The Taito-shell computing environment is part of the Taito cluster.

Taito-shell emulates normal application server like behavior. For the user, it appears very much like a normal Linux server: you can log in and run applications interactively with Linux commands. Taito-shell uses the same the disk environment, software stackand module system as the normal Taito cluster. The difference to the Taito login nodes is, that Taito-shell does not have the 1 hour time limit for jobs, that are executed interactively. Instead, a job started in Taito-shell can run as long as the Taito-shell session remains open.

From the technical point of view, Taito-shell is an oversubscribed interactive batch queue on the Taito super cluster. When you log in to *taito-shell.csc.fi*, you are automatically connected to an interactive batch job running in this unlimited partition.

Using this kind of approach has the following advantages over a traditional application server:

- Taito-shell uses the same resources as Taito, like home directory, $WRKDIR directory and installed software. You can easily switch between interactive working and batch jobs based working.
- Better scalability: In case of high demand, more Taito nodes can be assigned to Taito-shell and vice versa.
- Improved load balancing: The number of users per Taito-shell node can be better controlled.

Taito-shell is, however, not a separate Linux server and it has two important exceptions compared to traditional Linux application servers:

- **No direct node access**. "ssh taito-shell.csc.fi" logs in to the next free Taito-shell node on Taito. You cannot determine on which node you will end up. We are aware that specific node access is sometimes necessary, please see the FAQ section below on how to achieve this.
- **screen/nohup does not work** in Taito-shell. Logging out of Taito-shell kills all processes of the user in question, also background jobs. For long running jobs you can use Taito's batch job system, or, for smaller scale jobs, consult the FAQ.

The features of Taito-Shell environment in a nutshell:

- Direct login with the address: taito-shell.csc.fi
- Offers no-queuing interactive use of software installed on Taito
- Uses the same file system and network shares as Taito.
- Enables seamless switching between quick interactive use and batch jobs.
- Computing capacity up to 4 cores per user
- Shared memory of up to 128 GB per user

Jobs that utilize more than 4 cores or that require more than 128 GB memory usage should be executed as batch jobs in Taito or in Sisu.

7.2 Using Taito-shell

7.2.1 Obtaining a user id

If you have a user account for Taito, you can use it to login to *Taito-shell* too.

If you are a new customer for CSC's computing and application software services, see first instructions for new customers below. The application procedure depends on whether your are an academic user or not.

https://research.csc.fi/csc-guide-getting-access-to-csc-services

7.2.2 Logging in

There are two ways to login:shell login (ssh, Putty etc.) and *NoMachine* for graphical logins.

7.2.2.1 Logging in via ssh

To log in, normally one has to first access a local workstation (at university or some other site on the Internet) and then use an SSH program to connect to Taito-shell (**taito-shell.csc.fi**) with your **CSC user_id**:

```
ssh -X csc_user_id@taito-shell.csc.fi
```

After you have provided your password, you will see a prompt similar to the one below, the number might be different:

```
[user_id@c305 ~]
```

Taito-shell is now ready to execute your commands. For more information on connecting to CSC's servers via SSH please consult the general guide on how to use SSH at CSC.

7.2.2.2. Logging in via NoMachine

Graphical logins to *Taito-shell* are possible via NoMachine. You need to download and install a NoMachine client and configure it to use the server *nxkajaani.csc.fi*. Please consult the NoMachine documentation on how to get graphical access. To start Taito-Shell in *NoMachine*, right click with the mouse in the Desktop area, choose *Taito-shell* from the pull-down menu, add your password, and you are logged in. You can now type in the commands to launch the software you want to use, eg. for Rstudio commands would be

```
module load r-env
module load rstudio
rstudio
```

7.2.3 Submitting batch jobs from Taito-shell

Taito-shell.csc.fi environment is configured for running commands interactively. You can submit batch jobs from Taito-shell to taito.csc.fi cluster. However to do that, you need to add definition **-M csc** either top the batch job commands or to the #SBATCH lines of the batch job files.

For example to submit a batch job command you should use command:

```
sbatch -M csc batch_job_file.sh
```

In the same way, the status of the jobs , running in Taito cluster, can be checked with command

```
squeue -M csc -l
```

(Running *squeue* command without *-M csc* would list the current taito-shell.csc.fi sessions)

7.3 Taito-shell FAQ

7. Using Taito-shell for running interactive jobs in Taito

7.1 What is Taito-shell

The *Taito-shell* environment is intended for **interactive use of scientific applications maintained by CSC**, **serial jobs** and usage of **graphical application interfaces**. The Taito-shell computing environment is part of the Taito cluster.

Taito-shell emulates normal application server like behavior. For the user, it appears very much like a normal Linux server: you can log in and run applications interactively with Linux commands. Taito-shell uses the same the disk environment, software stackand module system as the normal Taito cluster. The difference to the Taito login nodes is, that Taito-shell does not have the 1 hour time limit for jobs, that are executed interactively. Instead, a job started in Taito-shell can run as long as the Taito-shell session remains open.

From the technical point of view, Taito-shell is an oversubscribed interactive batch queue on the Taito super cluster. When you log in to *taito-shell.csc.fi*, you are automatically connected to an interactive batch job running in this unlimited partition.

Using this kind of approach has the following advantages over a traditional application server:

- Taito-shell uses the same resources as Taito, like home directory, $WRKDIR directory and installed software. You can easily switch between interactive working and batch jobs based working.
- Better scalability: In case of high demand, more Taito nodes can be assigned to Taito-shell and vice versa.
- Improved load balancing: The number of users per Taito-shell node can be better controlled.

Taito-shell is, however, not a separate Linux server and it has two important exceptions compared to traditional Linux application servers:

- **No direct node access**. "ssh taito-shell.csc.fi" logs in to the next free Taito-shell node on Taito. You cannot determine on which node you will end up. We are aware that specific node access is sometimes necessary, please see the FAQ section below on how to achieve this.
- **screen/nohup does not work** in Taito-shell. Logging out of Taito-shell kills all processes of the user in question, also background jobs. For long running jobs you can use Taito's batch job system, or, for smaller scale jobs, consult the FAQ.

The features of Taito-Shell environment in a nutshell:

- Direct login with the address: taito-shell.csc.fi
- Offers no-queuing interactive use of software installed on Taito
- Uses the same file system and network shares as Taito.
- Enables seamless switching between quick interactive use and batch jobs.
- Computing capacity up to 4 cores per user
- Shared memory of up to 128 GB per user

Jobs that utilize more than 4 cores or that require more than 128 GB memory usage should be executed as batch jobs in Taito or in Sisu.

7.2 Using Taito-shell

7.2.1 Obtaining a user id

If you have a user account for Taito, you can use it to login to *Taito-shell* too.

If you are a new customer for CSC's computing and application software services, see first instructions for new customers below. The application procedure depends on whether your are an academic user or not.

https://research.csc.fi/csc-guide-getting-access-to-csc-services

7.2.2 Logging in

There are two ways to login:shell login (ssh, Putty etc.) and *NoMachine* for graphical logins.

7.2.2.1 Logging in via ssh

To log in, normally one has to first access a local workstation (at university or some other site on the Internet) and then use an SSH program to connect to Taito-shell (**taito-shell.csc.fi**) with your **CSC user_id**:

```
ssh -X csc_user_id@taito-shell.csc.fi
```

After you have provided your password, you will see a prompt similar to the one below, the number might be different:

```
[user_id@c305 ~]
```

Taito-shell is now ready to execute your commands. For more information on connecting to CSC's servers via SSH please consult the general guide on how to use SSH at CSC.

7.2.2.2. Logging in via NoMachine

Graphical logins to *Taito-shell* are possible via NoMachine. You need to download and install a NoMachine client and configure it to use the server *nxkajaani.csc.fi*. Please consult the NoMachine documentation on how to get graphical access. To start Taito-Shell in *NoMachine*, right click with the mouse in the Desktop area, choose *Taito-shell* from the pull-down menu, add your password, and you are logged in. You can now type in the commands to launch the software you want to use, eg. for Rstudio commands would be

```
module load r-env
module load rstudio
rstudio
```

7.2.3 Submitting batch jobs from Taito-shell

Taito-shell.csc.fi environment is configured for running commands interactively. You can submit batch jobs from Taito-shell to taito.csc.fi cluster. However to do that, you need to add definition **-M csc** either top the batch job commands or to the #SBATCH lines of the batch job files.

For example to submit a batch job command you should use command:

```
sbatch -M csc batch_job_file.sh
```

In the same way, the status of the jobs , running in Taito cluster, can be checked with command

```
squeue -M csc -l
```

(Running *squeue* command without *-M csc* would list the current taito-shell.csc.fi sessions)

7.3 Taito-shell FAQ

7. Using Taito-shell for running interactive jobs in Taito

7.1 What is Taito-shell

The *Taito-shell* environment is intended for **interactive use of scientific applications maintained by CSC**, **serial jobs** and usage of **graphical application interfaces**. The Taito-shell computing environment is part of the Taito cluster.

Taito-shell emulates normal application server like behavior. For the user, it appears very much like a normal Linux server: you can log in and run applications interactively with Linux commands. Taito-shell uses the same the disk environment, software stackand module system as the normal Taito cluster. The difference

to the Taito login nodes is, that Taito-shell does not have the 1 hour time limit for jobs, that are executed interactively. Instead, a job started in Taito-shell can run as long as the Taito-shell session remains open.

From the technical point of view, Taito-shell is an oversubscribed interactive batch queue on the Taito super cluster. When you log in to *taito-shell.csc.fi*, you are automatically connected to an interactive batch job running in this unlimited partition.

Using this kind of approach has the following advantages over a traditional application server:

- Taito-shell uses the same resources as Taito, like home directory, $WRKDIR directory and installed software. You can easily switch between interactive working and batch jobs based working.
- Better scalability: In case of high demand, more Taito nodes can be assigned to Taito-shell and vice versa.
- Improved load balancing: The number of users per Taito-shell node can be better controlled.

Taito-shell is, however, not a separate Linux server and it has two important exceptions compared to traditional Linux application servers:

- **No direct node access**. "ssh taito-shell.csc.fi" logs in to the next free Taito-shell node on Taito. You cannot determine on which node you will end up. We are aware that specific node access is sometimes necessary, please see the FAQ section below on how to achieve this.
- **screen/nohup does not work** in Taito-shell. Logging out of Taito-shell kills all processes of the user in question, also background jobs. For long running jobs you can use Taito's batch job system, or, for smaller scale jobs, consult the FAQ.

The features of Taito-Shell environment in a nutshell:

- Direct login with the address: taito-shell.csc.fi
- Offers no-queuing interactive use of software installed on Taito
- Uses the same file system and network shares as Taito.
- Enables seamless switching between quick interactive use and batch jobs.
- Computing capacity up to 4 cores per user
- Shared memory of up to 128 GB per user

Jobs that utilize more than 4 cores or that require more than 128 GB memory usage should be executed as batch jobs in Taito or in Sisu.

7.2 Using Taito-shell

7.2.1 Obtaining a user id

If you have a user account for Taito, you can use it to login to *Taito-shell* too.

If you are a new customer for CSC's computing and application software services, see first instructions for new customers below. The application procedure depends on whether your are an academic user or not.

https://research.csc.fi/csc-guide-getting-access-to-csc-services

7.2.2 Logging in

There are two ways to login:shell login (ssh, Putty etc.) and *NoMachine* for graphical logins.

7.2.2.1 Logging in via ssh

To log in, normally one has to first access a local workstation (at university or some other site on the Internet) and then use an SSH program to connect to Taito-shell (**taito-shell.csc.fi**) with your **CSC user_id**:

```
ssh -X csc_user_id@taito-shell.csc.fi
```

After you have provided your password, you will see a prompt similar to the one below, the number might be different:

```
[user_id@c305 ~]
```

Taito-shell is now ready to execute your commands. For more information on connecting to CSC's servers via SSH please consult the general guide on how to use SSH at CSC.

7.2.2.2. Logging in via NoMachine

Graphical logins to *Taito-shell* are possible via NoMachine. You need to download and install a NoMachine client and configure it to use the server *nxkajaani.csc.fi*. Please consult the NoMachine documentation on how to get graphical access. To start Taito-Shell in *NoMachine*, right click with the mouse in the Desktop area, choose *Taito-shell* from the pull-down menu, add your password, and you are logged in. You can now type in the commands to launch the software you want to use, eg. for Rstudio commands would be

```
module load r-env
module load rstudio
rstudio
```

7.2.3 Submitting batch jobs from Taito-shell

Taito-shell.csc.fi environment is configured for running commands interactively. You can submit batch jobs from Taito-shell to taito.csc.fi cluster. However to do that, you need to add definition **-M csc** either top the batch job commands or to the #SBATCH lines of the batch job files.

For example to submit a batch job command you should use command:

```
sbatch -M csc batch_job_file.sh
```

In the same way, the status of the jobs , running in Taito cluster, can be checked with command

```
squeue -M csc -l
```

(Running *squeue* command without *-M csc* would list the current taito-shell.csc.fi sessions)

7. Using Taito-shell for running interactive jobs in Taito

7.1 What is Taito-shell

The *Taito-shell* environment is intended for **interactive use of scientific applications maintained by CSC**, **serial jobs** and usage of **graphical application interfaces**. The Taito-shell computing environment is part of the Taito cluster.

Taito-shell emulates normal application server like behavior. For the user, it appears very much like a normal Linux server: you can log in and run applications interactively with Linux commands. Taito-shell uses the same the disk environment, software stackand module system as the normal Taito cluster. The difference to the Taito login nodes is, that Taito-shell does not have the 1 hour time limit for jobs, that are executed interactively. Instead, a job started in Taito-shell can run as long as the Taito-shell session remains open.

From the technical point of view, Taito-shell is an oversubscribed interactive batch queue on the Taito super cluster. When you log in to *taito-shell.csc.fi*, you are automatically connected to an interactive batch job running in this unlimited partition.

Using this kind of approach has the following advantages over a traditional application server:

- Taito-shell uses the same resources as Taito, like home directory, $WRKDIR directory and installed software. You can easily switch between interactive working and batch jobs based working.
- Better scalability: In case of high demand, more Taito nodes can be assigned to Taito-shell and vice versa.
- Improved load balancing: The number of users per Taito-shell node can be better controlled.

Taito-shell is, however, not a separate Linux server and it has two important exceptions compared to traditional Linux application servers:

- **No direct node access**. "ssh taito-shell.csc.fi" logs in to the next free Taito-shell node on Taito. You cannot determine on which node you will end up. We are aware that specific node access is sometimes necessary, please see the FAQ section below on how to achieve this.
- **screen/nohup does not work** in Taito-shell. Logging out of Taito-shell kills all processes of the user in question, also background jobs. For long running jobs you can use Taito's batch job system, or, for smaller scale jobs, consult the FAQ.

The features of Taito-Shell environment in a nutshell:

- Direct login with the address: taito-shell.csc.fi
- Offers no-queuing interactive use of software installed on Taito
- Uses the same file system and network shares as Taito.
- Enables seamless switching between quick interactive use and batch jobs.
- Computing capacity up to 4 cores per user
- Shared memory of up to 128 GB per user

Jobs that utilize more than 4 cores or that require more than 128 GB memory usage should be executed as batch jobs in Taito or in Sisu.

7.2 Using Taito-shell

7.2.1 Obtaining a user id

If you have a user account for Taito, you can use it to login to *Taito-shell* too.

If you are a new customer for CSC's computing and application software services, see first instructions for new customers below. The application procedure depends on whether your are an academic user or not.

https://research.csc.fi/csc-guide-getting-access-to-csc-services

7.2.2 Logging in

There are two ways to login:shell login (ssh, Putty etc.) and *NoMachine* for graphical logins.

7.2.2.1 Logging in via ssh

To log in, normally one has to first access a local workstation (at university or some other site on the Internet) and then use an SSH program to connect to Taito-shell (**taito-shell.csc.fi**) with your **CSC user_id**:

```
ssh -X csc_user_id@taito-shell.csc.fi
```

After you have provided your password, you will see a prompt similar to the one below, the number might be different:

```
[user_id@c305 ~]
```

Taito-shell is now ready to execute your commands. For more information on connecting to CSC's servers via SSH please consult the general guide on how to use SSH at CSC.

7.2.2.2. Logging in via NoMachine

Graphical logins to *Taito-shell* are possible via NoMachine. You need to download and install a NoMachine client and configure it to use the server *nxkajaani.csc.fi*. Please consult the NoMachine documentation on how to get graphical access. To start Taito-Shell in *NoMachine*, right click with the mouse in the Desktop area, choose *Taito-shell* from the pull-down menu, add your password, and you are logged in. You can now type in the commands to launch the software you want to use, eg. for Rstudio commands would be

```
module load r-env
module load rstudio
rstudio
```

7.2.3 Submitting batch jobs from Taito-shell

Taito-shell.csc.fi environment is configured for running commands interactively. You can submit batch jobs from Taito-shell to taito.csc.fi cluster. However to do that, you need to add definition **-M csc** either top the batch job commands or to the #SBATCH lines of the batch job files.

For example to submit a batch job command you should use command:

```
sbatch -M csc batch_job_file.sh
```

In the same way, the status of the jobs , running in Taito cluster, can be checked with command

```
squeue -M csc -l
```

(Running *squeue* command without *-M csc* would list the current taito-shell.csc.fi sessions)

7.3 Taito-shell FAQ