Très Grand Centre de calcul du CEA

TGCC public Documentation

Release 2025-05-05.1501

CEA

2025-05-05

CONTENT

INTRODUCTION

1.1 The computing center

The **TGCC** (Très Grand Centre de Calcul du CEA) is a high performance computing infrastructure aiming at hosting state-of-the-art supercomputers in France.

It is designed to:

- Accommodate future high end computing systems.

- Provide a communication and exhibition space for scientific events (conferences, seminars, training sessions, …).

- Propose a flexible and modular facility for future evolution of HPC systems.

At the moment, the TGCC hosts Joliot-Curie, a 21 PFlops supercomputer that is part of **GENCI** (Grand Equipement National de Calcul Intensif). It represents the french contribution to the European **PRACE** (Partnership for Advanced Computing in Europe) infrastructure.

The TGCC also hosts the **CCRT** (Centre de Calcul Recherche et Technologie, used by the CEA and its industrial partners). A full and up to date list of the CCRT industrial partners can be found on the ccrt website. The CCRT also provides the data storage and processing infrastructure for the France Génomique community, providing the highest level of competitiveness and performance in the production and analysis of genomic data.

The computing center architecture is data-centric. The computing nodes are connected to the private Lustre storage system for very fast I/O. And Global Lustre filesystems are shared between the different supercomputers. A hierarchical data storage system manages petabytes of data and is used for long term storage and archiving of results.

The computing center operates round the clock, except for scheduled maintenance periods when the teams update the hardware, firmware and software. The regular maintenance periods also allow a check of the general behavior and performance of the various supercomputers. Every year, a compulsory electrical maintenance requires a full shut down of the supercomputers for a longer period (3 days).

On-site support for the contractors, expert administration teams and an on-call duty system optimize the availability of the computing service. Guaranteeing access security and data confidentiality is a major preoccupation. A CEA IT security experts unit controls a security supervision system which monitors, detects and analyzes security alerts, enabling the security managers to react extremely rapidly.

Note: The Hotline is the single point of contact for any question or support request:

- e-mail : please refer to internal documentation to get hotline email

- phone : please refer to internal documentation to get hotline phone

Depending on its type, the demand will be transmitted to the appropriate team.

A users committee (COMUT) takes place every trimester to exchange between users and the TGCC staff. The representatives of each community can provide their feedback on the general usage of the computing center. Any big change or update is announced during the COMUT.

Training sessions are organized by CCRT staff on a regular basis.

1.2 Access to computing resources

Computing hours on the computing center are granted in a community-dependent fashion:

- **CCRT** Partners automatically get a share of the computing resources.

- French research teams can ask for resources through **GENCI** thanks to **DARI** calls: http://www.edari.fr

- Scientists and researchers from academia and industry can ask for resources through **PRACE**. PRACE accesses are of 2 kinds:

    - Preparatory Access is intended for short-term access to resources, for code-enabling and porting, required to prepare proposals for Project Access and to demonstrate the scalability of codes. Applications for Preparatory Access are accepted at any time, with a cut-off date every 3 months.

    - Project Access is intended for individual researchers and research groups including multinational research groups. It can be used for 1-year production runs, as well as for 2-year or 3-year (Multi-Year Access) production runs.

    Project Access is subject to the PRACE Peer Review Process, which includes technical and scientific review. Technical experts and leading scientists evaluate the proposals submitted in response to the bi-annual calls. Applications for Preparatory Access undergo technical review only. For more information on how to apply for access to PRACE resources, go to http://www.prace-ri.eu/how-to-apply

Ongoing PRACE projects are monitored as much as possible. Therefore, we regularly ask for your feedback. Feel free to tell us about any issue you are facing.

Project or partner accounts are granted an amount of computing hours or a computing share. They must use the awarded hours on a regular basis. To ensure that: over-consumption lowers priority so jobs might need more time to access resources, and reaching an under-consumption limit may result in hours being removed from a project.

TWO

GETTING STARTED

Welcome to your journey into CEA's high-performance computing! This guide will help you learn the basics of working with CEA's supercomputers. It covers job execution, module management, and the effective use of storage systems and services.

2.1 Accessing the HPC System

To start utilizing our supercomputer, first establish an SSH connection as follows:

```
$ ssh <login>@|fdqn|
```

For more detailed instructions on accessing the system, please see the *Interactive access section*.

2.2 Setting up Your User Environment

Upon connecting to the supercomputer, you have the opportunity to establish your user environment using the *Environment-Modules* tool.

Modules is a tool that allows you to control the software packages in your environment. It enables users to dynamically adjust their environment for their specific needs at any given time, without conflicting with other uses.

To see a list of all available software modules, you can use the **module avail** command:

```
$ module avail
```

To check which modules are currently loaded in your environment, use the **module list** command:

```
$ module list
```

The most common task is loading a module, which sets up your environment to use a specific software package. To load a module, use the **module load** command followed by the name of the software package:

```
$ module load python3
$ module load mpi
$ module load cuda
...
```

Once you have set up a suite of modules that fits your workflow, we recommend saving this setup using the **module save** command:

```
$ module save
```

With this, your module configuration will persist across different SSH connections, providing a consistent working environment.

You can unload a module with `module unload <module_name>` command or purge them all with `module purge`.

For more detailed information on using software modules and setting up your environment, please refer to the *Environment management section* of this guide.

2.3 Compiling Code on the Cluster

Our cluster provides a variety of compilers through the module system, such as GCC, Intel, and NVHPC.

For instance, to compile a Fortran program with MPI using GCC (Gnu Compiler Collection) you can use the following commands:

```
$ module load gcc
$ module load mpi
$ mpifort -o program_name program.f90
```

For those wanting to use other programming languages,compilers, or want more details about developping and compiling on the cluster, please see the *Parallel programming section* .

2.4 Using Python on the Cluster

Our cluster supports several Python distributions for different uses described in *Python section*.

For instance, for Python programming with parallel processing using MPI (includes mpi4py):

```
$ module load mpi python3
```

Once the appropriate module is loaded, you can start a Python interpreter by simply typing `python` in your terminal, or run a Python script using `python script_name.py`.

```
$ module load mpi python3
$ python script.py
```

To list all available Python packages in your current environment, run `pip3 list`.

For a more detailed guide on using Python on our cluster, including Machine learning and AI, refer to the *Python section*.

2.5 Submitting Your First Job

Working with our supercomputer involves job submissions. The login node serves for preparatory tasks and is not meant for calculations. Instead, computationally intensive tasks, even Python scripts, should run on compute nodes.

Job submission involves creating a script and then submitting this to the job scheduler. This approach ensures efficient usage and fair access to computational resources.

Creating a Job Script

A job script is a simple text file containing directives for the job scheduler and the commands you want to run.

Here is basic example using MPI:

```
$ nano job_mpi.sh
```

Inside the editor:

```
#!/bin/bash
#MSUB -r my_job_mpi          # Job name
#MSUB -n 32                  # Number of tasks to use
#MSUB -c 1                   # Number of cores (or threads) per task to use
#MSUB -T 1800                # Elapsed time limit in seconds of the job (default:␣
→7200
#MSUB -o my_job_mpi_%I.o     # Standard output. %I is the job id
#MSUB -e my_job_mpi_%I.e     # Error output. %I is the job id
#MSUB -A <project>           # Project ID
#MSUB -q |default_CPU_partition| # Partition name (see ccc_mpinfo)
set -x
cd ${BRIDGE_MSUB_PWD}
ccc_mprun ./a.out
```

For more job scripts examples (OpenMP, CUDA...), please refer to the *job submission scripts examples section*.

Submitting the Job

Once your job script is ready, you can submit it to the scheduler:

```
$ ccc_msub job_mpi.sh
```

For small tests you may use an interactive session as described in *the interactive submission section*.

For a more detailed guide on creating job scripts and submitting jobs, refer to the *Job subsmission section*.

2.6 Understanding Data Spaces

Our supercomputer offers multiple file systems, each optimized for different use cases.

Remember, choosing the right file system for your needs can significantly impact the performance and efficiency of your jobs. For more details, look at the *Data spaces section*.

HOME

The HOME directory is best used for storing small, essential files. Its space and performance are limited, and it is subject to snapshots for backup. Avoid storing large datasets here.

It is accessible through the `$CCCHOME` variable.

This space is always available.

SCRATCH

The SCRATCH space offers high performance and is designed for temporary storage during your jobs. Files in SCRATCH are purged every 60 days, so remember to move important data to a more permanent location.

It is accessible through the `$CCCSCRATCHDIR` variable.

By default this space is unavailable for your jobs. You need to have `#MSUB -m scratch` at the beggining of your submission script to use it.

WORK

The WORK space is similar to SCRATCH but offers lower performance. It is not subject to purging, making it suitable for installing products, storing documents, and holding work-in-progress data.

It is accessible through the `$CCCWORKDIR` variable.

By default this space is unavailable for your jobs. You need to have the `#MSUB -m work` at the beggining of your submission script to use it.

STORE

The STORE space provides extensive storage for archives, old results, and other large data that you do not need to access frequently.

Be aware that data in STORE may migrate to magnetic tapes over time. Retrieving this data can take between 20-40 minutes. For managing such scenarios, please refer to the *Data Management on STORE section*.

It is accessible through the `$CCCSTOREDIR` variable.

By default this file system is unavailable for your jobs. You need to the `#MSUB -m store` at the beggining of your submission script to use it.

TMP

The TMP is a local, non-shared file system on each node. It is ideal for transient files used in minor operations.

It is accessible through the `$TMPDIR` variable.

2.7 Further Topics

Beyond the basics, our supercomputer supports advanced features that can enhance your computational work:

Debugging & Profiling

Optimize your code with debugging and profiling. See the *Debugging section* and *Profiling section* for details.

GPU Programming

Utilize our GPU accelerators to boost your work. Refer to the *GPU-accelerated computing section* for more.

Containers & Virtualization

Achieve portable and reproducible environments with containers and virtualization. Details in the Virtualization and containers section.

Quantum Computing

Explore new frontiers with quantum computing. Visit the *Quantum section* for more information.

Visualization Services

Our remote desktop service allow for fast and high-quality visualization of your results. Check out the *Interactive access section* for more.

TRAININGS

Computing center staff regularly organize training sessions, whose content remains accessible outside these sessions.

The training material, slides and lab files, are available as modules, just like the other applications.

```
$ module avail formation
```

Once you have found your training, you can load the module the same way you did in the actual session.

```
$ module load formation/base # Or any training you are interested in
```

Once you load the module, you'll be able to access and copy the materials into any directory you choose.

```
$ cp -r $FORMATION_LABS <your labs folder>
$ cp -r $FORMATION_SLIDES <your training slides folder>
$ module unload formation
```

> **Warning:** Do not forget to unload the formation module before submitting outside training sessions. Otherwise, your job won't be able to start.

FOUR

SUPERCOMPUTER ARCHITECTURE

4.1 Configuration of Irene

The compute nodes are gathered in partitions according to their hardware characteristics (CPU architecture, amount of RAM, presence of GPU, etc). A partition is a set of identical nodes that can be targeted to host one or several jobs. Choosing the right partition for a job depends on code prerequisites in term of hardware resources. For example, executing a code designed to be GPU accelerated requires a partition with GPU nodes.

The Irene supercomputer offers three different kind of nodes: regular compute nodes, large memory nodes, and GPU nodes.

- **Skylake nodes for regular computation**

    - Partition name: skylake

    - CPU : 2x24-cores Intel Skylake@2.7GHz (AVX512)

    - Cores/Node: 48

    - Nodes: 1 653

    - Total cores: 79 344

    - RAM/Node: 180GB

    - RAM/Core: 3.75GB

- **AMD Rome nodes for regular computation**

    - Partition name : Rome

    - CPUs: 2x64 AMD Rome@2.6Ghz (AVX2)

    - Core/Node: 128

    - Nodes: 2286

    - Total core: 292 608

    - RAM/Node: 228GB

    - RAM/core : 1.8GB

- **Hybrid nodes for GPU computing and graphical usage**

    - Partition name: hybrid

    - CPUs: 2x24-cores Intel Skylake@2.7GHz (AVX2)

    - GPUs: 1x Nvidia Pascal P100

    - Cores/Node: 48

- – Nodes: 20

- – Total cores: 960

- – RAM/Node: 180GB

- – RAM/Core: 3.75GB

- – I/O: 1 HDD 250 GB + 1 SSD 800 GB/NVMe

- **Fat nodes with a lot of shared memory for computation lasting a reasonable amount of time and using no more than one node**
  - – Partition name: xlarge

  - – CPUs: 4x28-cores Intel Skylake@2.1GHz

  - – GPUs: 1x Nvidia Pascal P100

  - – Cores/Node: 112

  - – Nodes: 5

  - – Total cores: 560

  - – RAM/Node: 3TB

  - – RAM/Core: 27GB

  - – IO: 2 HDD de 1 TB + 1 SSD 1600 GB/NVMe

- **V100 nodes for GPU computing and AI**
  - – Partition name: V100

  - – CPUs: 2x20-cores Intel Cascadelake@2.1GHz (AVX512)

  - – GPUs: 4x Nvidia Tesla V100

  - – Cores/Node: 40

  - – Nodes: 32

  - – Total cores: 1280 (+ 128 GPU)

  - – RAM/Node: 175 GB

  - – RAM/Core: 4.4 GB

- **V100l nodes for GPU computing and AI**
  - – Partition name: V100

  - – CPUs: 2x18-cores Intel Cascadelake@2.6GHz (AVX512)

  - – GPUs: 1x Nvidia Tesla V100

  - – Cores/Node: 36

  - – Nodes: 30

  - – Total cores: 1080 (+ 30 GPU)

  - – RAM/Node: 355 GB

  - – RAM/Core: 9.9 GB

- **V100xl nodes for GPU computing and AI**
  - – Partition name: V100

  - – CPUs: 4x18-cores Intel Cascadelake@2.6GHz (AVX512)

– GPUs: 1x Nvidia Tesla V100

– Cores/Node: 72

– Nodes: 2

– Total cores: 144 (+ 30 GPU)

– RAM/Node: 2.9 TB

– RAM/Core: 40 GB

Note that depending on the computing share owned by the partner you are attached to, you may not have access to all the partitions. You can check on which partition(s) your project has allocated hours thanks to the command `ccc_myproject`.

ccc_mpinfo displays the available partitions/queues that can be used on a job.

```
$ ccc_mpinfo
                       --------------CPUS------------   -------------NODES------------
PARTITION     STATUS    TOTAL   DOWN    USED    FREE    TOTAL   DOWN    USED    FREE     ␣
→MpC   CpN SpN CpS TpC
---------     ------   ------  ------  ------  ------   ------  ------  ------  ------    --
→--- --- --- --- ---
skylake       up         9960       0    9773     187      249       0     248       1    ␣
→4500   40   2   20   1
xlarge        up          192       0     192       0        3       0       3       0   ␣
→48000   64   4   16   1
hybrid        up          140       0      56      84        5       0       2       3   ␣
→8892   28   2   14   1
v100          up          120       0       0     120        3       0       0       3   ␣
→9100   40   2   20   1
```

- **MpC** : amount of memory per core
- **CpN** : number of cores per node
- **SpN** : number of sockets per node
- **Cps** : number of cores per socket
- **TpC** : number of threads per core This allows for SMT (Simultaneous Multithreading, as hyperthreading for Intel architecture)

4.2 Interconnect

The compute nodes are connected through a EDR InfiniBand network in a pruned FAT tree topology. This high through-put and low latency network is used for I/O and communications among nodes of the supercomputer.

4.3 Lustre

Lustre is a type of parallel distributed file system, commonly used for large-scale cluster computing. It actually relies on a set of multiple I/O servers and the Lustre software presents them as a single unified filesystem.

The major Lustre components are the MDS (MetaData Server) and OSSs (Object Storage Servers). The MDS stores metadata such as file names, directories, access permissions, and file layout. It is not actually involved in any I/O operations. The actual data is stored on the OSSs. Note that one single file can be stored on several OSSs which is one of the benefits of Lustre when working with large files.
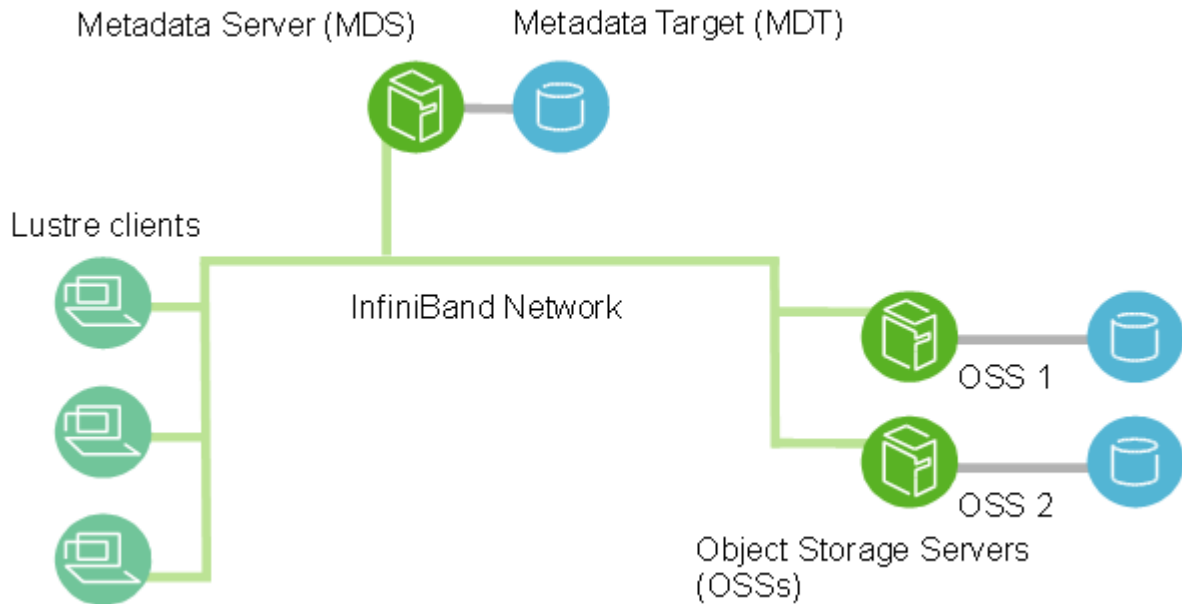
Fig. 1: Lustre

More information on how Lustre works and best practices are described in *Lustre best practice*.

FIVE

USER ACCOUNT

> **Warning:** Please refer to internal technical documentation to get information about this subject.

INTERACTIVE ACCESS

6.1 System access

> **Warning:** Please refer to internal technical documentation to get information about this subject.

6.2 Login nodes usage

When you connect to the supercomputer, you are directed to one of the login nodes of the machine. Many users are simultaneously connected to these login nodes. Since their resources are shared it is important to respect some standards of good practice.

6.2.1 Usage limit on login nodes

Login nodes should only be used to interact with the batch manager and to run lightweight tasks. As a rule of thumb, any process or group of processes which would use more CPU power and/or memory than what is available on a basic personal computer should not be executed on a login node. For more demanding interactive tasks, you should allocate dedicated resources on a compute node.

To ensure a satisfying experience for all users, offending tasks are automatically throttled or killed.

6.2.2 Interactive submission

There are 2 possible ways of accessing computing resources without using a submission script. `ccc_mprun -K` allows to create the allocation and the job environment while you are still on the login node. It is useful for MPI tests. `ccc_mprun -s` opens a shell on a compute node. It is useful for sequential or multi-threaded work that would be too costly for the login node.

Allocate resources interactively (-K)

It is possible to work interactively on allocated compute nodes thanks to the option –K of **ccc_mprun**.

```
$ ccc_mprun -p partition -n 8 -T 3600 -A <project> -K
```

This command will create an allocation and start a job.

```
$ echo $SLURM_JOBID
901732
```

Within this reservation, you can run job steps with **ccc_mprun**. Since the compute nodes are already allocated, you will not have to wait.

```
$ ccc_mprun hostname
node1569
node1569
node1569
node1569
node1569
node1569
node1569
node1569
```

Note: At this point, you are still connected on the login node. You cannot run your code directly or with mpirun. you have to use **ccc_mprun**!

Such an allocation is useful when developing an MPI program, in order to be able to test a **ccc_mprun** quickly and several times in a short period.

You can use the usual options for **ccc_mprun** as in *ccc_mprun options*.

Working on a compute node (-s)

To directly work on the allocated compute nodes, you need to open a shell inside a SLURM allocation. This is possible thanks to the -s option:

```
$ ccc_mprun -p partition -c 16 -m work,scratch -s
[node1569 ~]$ hostname
  node1569
```

In this case, the 16 cores of the node node1569 are allocated to you and you are able to run freely on all those cores without disrupting other jobs. It is just like any allocation, the computing hours will be accounted on your project. In any case, you will need to specify the filesystems you want to have access to, using -m option followed by a comma-separated list of filesystems.

Note:

- You cannot use multiple nodes in this mode. You are limited to the number of cores in one node.
- When you do not need it any longer, do not forget to stop the interactive session with ctrl+d or exit 0.
- The computing hours spent in the interactive session will be withdrawn from your project quota.
- You can wait for the allocation for a shorter time using the "test" QOS.

This is typically used for costly and punctual tasks such as compiling a large code on 16 cores, post-processing or aggregating output data etc.

At any point, you can use the **hostname** command to check whether you are on a compute node or on a login nodes.

You can use the usual options for **ccc_mprun** as in *ccc_mprun options*.

6.3 Remote Desktop System service (NiceDCV)

The Remote Desktop System service on the TGCC supercomputer is based on NiceDCV solution. NiceDCV provides a web site where you can launch the booked graphical session.

6.3.1 Book a session

There are two types of sessions : a "virtual" session and a "console" session.

- The virtual session is using the GNOME Display Manager (GDM) unlike the console session. The virtual session is by default launched on only one GPU and not on all the GPUs of the node. Therefore, it is possible to launch in parallel as many virtual sessions as the number of available GPUs on the node. Moreover, many users can launch a virtual session on the same node given that they are on the same container.

The following command allows to book a virtual session :

```
$ ccc_visu virtual -p partition
```

- The console session allows node exclusivity, i.e. all GPUs on the node can be used inside the session. The visualization performance could be faster than on the virtual session (almost 2 times faster).

The following command allows to book for a console session :

```
$ ccc_visu console -p partition
```

6.3.2 Session access

Once the reservation is booked from the above commands, a URL is given in the standard output. To access the graphical session, launch this URL on your browser. A page opens in which you are asked to give your Irene credentials. Then, to acess the allocated session, you have two options:

- The first client (on the left): opens the session inside the brower. You need to give again your Irene credentials.

- The second client (on the right): allows to download it. However, you need root access on your machine to be able to install it.

Moreover, while the allocation with command **ccc_visu** didn't expire or hadn't been cancelled, the session will not be impacted if the client is closed. A session will last as defined by the -T option (in seconds), or 2 hours by default. The graphical session will expire with the corresponding job, and will not be available afterward. Besides, the session can be closed from the terminal from which the session was started using **CTL-D** or **exit**, or by ending the corresponding job with the **ccc_mdel** command.

DATA SPACES

All user data are stored on file systems reachable from the supercomputer login and compute nodes. Each file system has a purpose: your HOME directory contains user configuration, your SCRATCH directory is intended for temporary computational data sets, etc. To prevent over-usage of the file system capacities, limitations/quotas are set on data space allocated to users or groups. Except for your HOME directory, which is hosted by a NFS file system, user data spaces are stored on Lustre file systems. This section provides data usage guidelines that must be followed, since an inappropriate usage of the file systems might badly affect the overall production of the computing center.

7.1 Available file systems

This section introduces the available file systems with their purpose, their quota policy and their recommended usage.

File System	*Personal*	*Shared*
Home	`$CCCHOME` `$OWN_HOME` `$CCFRHOME` `$OWN_CCFRHOME`	`$ALL_CCCHOME` `$ALL_CCFRHOME` `$<SHSPACE>_ALL_CCCHOME` `$<SHSPACE>_ALL_CCFRHOME` `$<SHSPACE>_ALL_HOME` `$OWN_ALL_CCCHOME` `$OWN_ALL_CCFRHOME` `$OWN_ALL_HOME`
Scratch	`$CCCSCRATCHDIR` `$OWN_CCCSCRATCHDIR` `$CCFRSCRATCH` `$OWN_CCFRSCRATCH`	`$ALL_CCCSCRATCHDIR` `$ALL_CCFRSCRATCH` `$<SHSPACE>_ALL_CCCSCRATCHDIR` `$<SHSPACE>_ALL_CCFRSCRATCH` `$OWN_ALL_CCCSCRATCHDIR` `$OWN_ALL_CCFRSCRATCH`
Work	`$CCCWORKDIR` `$OWN_CCCWORKDIR` `$CCFRWORK` `$OWN_CCFRWORK`	`$ALL_CCCWORKDIR` `$ALL_CCFRWORK` `$<SHSPACE>_ALL_CCCWORKDIR` `$<SHSPACE>_ALL_CCFRWORK` `$OWN_ALL_CCCWORKDIR` `$OWN_ALL_CCFRWORK`
Store	`$CCCSTOREDIR` `$OWN_CCCSTOREDIR` `$CCFRSTORE` `$OWN_CCFRSTORE`	`$ALL_CCCSTOREDIR` `$ALL_CCFRSTORE` `$<SHSPACE>_ALL_CCCSTOREDIR` `$<SHSPACE>_ALL_CCFRSTORE` `$OWN_ALL_CCCSTOREDIR` `$OWN_ALL_CCFRSTORE`

HOME

The HOME is a slow, small file system with backup that can be used from any machine.

Characteristics:

- Type: NFS
- Data transfer rate: low
- Quota: 5GB/user
- Usage: Sources, job submission scripts. . .
- Comments: Data are saved
- Access: from all resources of the center
- *Personnal variables*: `$CCCHOME $OWN_HOME $CCFRHOME $OWN_CCFRHOME`
- *Shared variables*: `$ALL_CCCHOME` `$ALL_CCFRHOME` `$<SHSPACE>_ALL_CCCHOME` `$<SHSPACE>_ALL_CCFRHOME` `$<SHSPACE>_ALL_HOME` `$OWN_ALL_CCCHOME` `$OWN_ALL_CCFRHOME` `$OWN_ALL_HOME`

Note:

- HOME is the only file system without limitation on the number of files (quota on inodes)
- The retention time for HOME directories backup is 6 months
- The backup files are under the `~/.snapshot` directory

SCRATCH

The SCRATCH is a very fast, big and automatically purged file system.

Characteristics:

- Type: Lustre
- Data transfer rate: 300 GB/s
- Quota: The quota is defined by group. The command `ccc_quota` provides information about the quota of the groups you belong to. By default, a quota of 2 millions inodes and 100 To of disk space is granted for each data space.
- Usage: Data, Code output. . .
- Comments: This filesystem is subject to purge
- Access: Local to the supercomputer
- *Personnal variables*: `$CCCSCRATCHDIR $OWN_CCCSCRATCHDIR $CCFRSCRATCHDIR $OWN_CCFRSCRATCHDIR`
- *Shared variables*: `$ALL_CCCSCRATCHDIR $ALL_CCFRSCRATCHDIR $<SHSPACE>_ALL_CCCSCRATCHDIR` `$<SHSPACE>_ALL_CCFRSCRATCH $OWN_ALL_CCCSCRATCHDIR $OWN_ALL_CCFRSCRATCH`

The purge policy is as follows:

- Files not accessed for 60 days are **automatically** purged
- Symbolic links are not purged
- Directories that have been empty for more than 30 days are removed

Note:

- You may use the `ccc_will_purge` command to display the files to be purged soon. Please read the dedicated section for more information

WORK

WORK is a fast, medium and permanent file system (but without backup):

Characteristics:

- Type: Lustre via routers

- Data transfer rate: high (70 GB/s)

- Quota: 5 TB and 500 000 files/group

- Usage: Commonly used file (Source code, Binary...)

- Comments: Neither purged nor saved (tar your important data to STORE)

- Access: from all resources of the center

- *Personnal variables*: $CCCWORKDIR $OWN_CCCWORKDIR $CCFRWORK $OWN_CCFRWORK

- *Shared variables*: $ALL_CCCWORKDIR $ALL_CCFRWORK $<SHSPACE>_ALL_CCCWORKDIR $<SHSPACE>_ALL_CCFRWORK $OWN_ALL_CCCWORKDIR $OWN_ALL_CCFRWORK

Note:

- WORK is smaller than SCRATCH, it's only managed through quota.

- This space is not purged but **not saved** (regularly backup your important data as tar files in STORE)

STORE

STORE is a huge storage file system

Characteristics:

- Type: Lustre + HSM

- Data transfer rate: high (200 GB/s)

- Quota: 50 000 files/group, expected file size range 10GB-1TB

- Usage: To store of large files (direct computation allowed in that case) or packed data (tar files...)

- Comments: Migration to hsm relies on file modification time: avoid using **cp** options like -p, -a ...

- Access: from all resources of the center

- *Personnal variables*: $CCCSTOREDIR $OWN_CCCSTOREDIR $CCFRSTORE $OWN_CCFRSTORE

- *Shared variables*: $ALL_CCCSTOREDIR $ALL_CCFRSTORE $<SHSPACE>_ALL_CCCSTOREDIR $<SHSPACE>_ALL_CCFRSTORE $OWN_ALL_CCCSTOREDIR $OWN_ALL_CCFRSTORE

- Additional info: Use **ccc_hsm status <file>** to know whether a file is on disk or tape level, and **ccc_hsm get <file>** to preload file from tape before a computation

Note:

- STORE has no limit on the disk usage (quota on space)

- STORE usage is monitored by a scoring system

- An HSM (Hierarchical Storage Management) is a data storage system which automatically moves data between high-speed and low-speed storage media. In our case, the high speed device is a Lustre file system and the low speed device consists of magnetic tape drives. Data copied to the HSM filesystem will be moved to magnetic

tapes (usually depending on the modification date of the files). Once the data is stored on tape, accessing it will be slower.

- To fetch and store big data volumes on the Store, you may use the **ccc_pack** command. Please see the **dedicated subsection** for more information.

TMP

TMP is a local, fast file system but of limited size.

Characteristics:

- Type: zram (RAM)

- Data transfer rate: very high (>1GB/s) and very low latency

- Size: 16 GB

- Usage: Temporary files during a job

- Comments: Purge after each job

- Access: Local within each node

- Variable: `CCCTMPDIR` or `CCFRTMP`

Note:

- TMP allow fast write and read for local needs.

- TMP is local to the node. Only jobs/processes within the same node have access to the same files.

- Write files in TMP will reduce the amount of available RAM.

SHM

SHM is a very fast, local file system but of limited size.

Characteristics:

- Type: tmpfs (RAM, block size 4Ko)

- Data transfer rate: very high (>1GB/s) and very low latency

- Size: 50% of RAM

- Usage: Temporary files during compute; can be used as a large cache

- Comments: Purge after each job

- Access: Local within each node

- Variable: `CCCSHMDIR`

Note:

- SHM allow very fast file access.

- SHM is local to the node. Only jobs/processes within the same node have access to the same files.

- Write files in SHM will reduce the amount of available RAM.

- Using the SHM folder through the CCCSHMDIR variable is strongly recommended, especially for small files.

> **Warning:** CCCSHMDIR is only available during a SLURM allocation (ccc_msub). CCCSHMDIR is not available in an interactive session (ccc_mprun -s).

> **Warning:** Using CCCSHMDIR is necessary in jobs to make sure data are properly cleaned at job end. It may cause node deadlocks otherwise.

7.2 Quota

Because file systems are shared between many users, restrictions are enforced on disk usage. The disk quotas and your current usage (space and number of files or directories) are shown by the command **ccc_quota**. This command also displays a score rating your usage on Lustre + HSM file systems ("Account scoring").

```
$ ccc_quota
Disk quotas for user <login> (uid <uid>):

          ============ SPACE ============   ============ INODE ============
 Filesystem    usage    soft    hard   grace    entries    soft    hard   grace
 ----------  -------- ------- ------- -------   -------- ------- ------- -------
       HOME    2.34G      5G      5G       -      48.01k       -       -       -
    SCRATCH   78.96G       -       -       -       1.19M       -       -       -
       WORK    36.7G       -       -       -      13.13k       -       -       -
      STORE      40k       -       -       -          10       -       -       -

Disk quotas for data space <group> (gid <gid>):

          ============ SPACE ============   ============ INODE ============
 Filesystem    usage    soft    hard   grace    entries    soft    hard   grace
 ----------  -------- ------- ------- -------   -------- ------- ------- -------
       HOME        -       -       -       -           -       -       -       -
    SCRATCH    2.01T    100T  100.1T       -       1.24M      2M   2.05M       -
       WORK    6.08T     10T     10T       -       3.13M      5M      5M       -
      STORE  104.13M       -       -       -       7.73k     50k   50.5k       -

STORE filesystem scoring for group <group> (gid <gid>):

 ================ SPACE ================   ============== INODE ==============
    usage files<500M  files<1G  avg_fsize    entries     dir  symlink  non_files
 -------- ---------- --------- ----------   -------- ------- -------- ----------
   10.07T     69.53%    69.58%      1.48G      7.73k     755        0      9.76%
           ---------- --------- ----------                        ----------
                1/7 +     1/3 +      3/4                                 +     6/6
           ================================ TOTAL SCORE: 11/20 =============
```

7.2.1 Disk quotas

Three parameters govern the disk quotas:

- *soft limit*: when your usage reaches this limit you will be warned.

- *hard limit*: when your usage reaches this limit you will be unable to write new files.

- *grace period*: the period during which you can exceed the *soft limit*.

Within the computing center, the quotas have been defined as follows:

- the *grace period* is 1 week. It means that once you have reached the soft limit, a countdown starts for a week. By the end of the countdown, if you have not brought your quota under the soft limit, you will not be able to create new files anymore. Once you get back under the soft limit, the countdown is reset.

- *soft limit* and *hard limit* are almost always the same, which means that checking your current usage to be below this limit is enough.

- those limits have been set *on both the number of files (inode) and the data usage (space)* for *WORK* and *SCRATCH*.

- those limits have been set *only on the data usage (space)* for *HOME*.

- those limits have been set *only on the number of files (inode)* for *Lustre + HSM file systems (CCCSTOREDIR)*.

- Lustre + HSM file systems (CCCSTOREDIR) have a scoring system instead of space limitations.

Note: The section *Disk quotas* displayed by `ccc_quota` is updated in real-time.

7.2.2 Account scoring

On Lustre + HSM file systems (CCCSTOREDIR), a score reflects how close you are from the recommended usage.

4 criteria are defined and are each granted a certain amount of points. Here are those criteria:

- inodes should be regular files (not directories nor symbolic links) for 6 points. With limits of 10%, 25%, 20%, 30%, and 50%, you will lose one point for each limit exceeded. With a percentage of 20.74% "non_files" you will have a subscore of 3 points;

- files should be bigger than 500MB for 7 points. With limits of 10%, 20%, 30%, 40%, 50%, 60%, and 70%, you will lose one point for each limit exceeded. With 67.20% of small files (<500M) you will have a subscore of 1;

- files should be bigger than 1GB for 3 points. With limits of 25%, 60%, and 75%, you will lose one point for each limit exceeded;

- the average file size should be high enough for 4 points. With limits of 64MB, 128MB, 1GB, and 8GB, you will gain one point for each limit. With an average file size of 2GB you will have a subscore of 3 points out of 4.

The command `ccc_quota` shows how well you match the criteria in the *Account Scoring* section:

- current usage (space and inodes)

- percentages of files not matching the criteria

- global score and the subscores.

Here is an example of scoring:

```
STORE filesystem scoring for group <group> (gid <gid>):

 =============== SPACE ===============   ============= INODE =============
    usage files<500M  files<1G  avg_fsize   entries      dir  symlink  non_files
 --------  ---------  ---------  ---------  --------  -------  --------  ---------
  16.97T     92.09%     92.33%       626M    35.87k     6235      1204     20.74%
            ---------  ---------  ---------                                ---------
              0/7 +     0/3 +      2/4                             +       3/6
            =============================== TOTAL SCORE: 5/20 =============
```

- For instance, for the criteria "non_files", the score is of 3 out of 6

- The related number is 20.74%: this is the proportion of your inodes that are directories of links.

- These 20.74% represent three points lost out of 6

- The subscore of the criteria is computed and is then 3/6

- Finally all subscores are summed and constitute a global score

```
ex: (0+0+2+3)=5 points over (7+3+4+6)=20, that is to say 5/20
```

Note: The section *Account Scoring* displayed by **ccc_quota** is updated daily (around 13:00 CET/CEST).

7.3 Data protection

Warning: Please refer to internal technical documentation to get information about this subject.

7.4 Personal Spaces

Every user of the computing center has a personal space in their primary work group, on each available filesystems: NFS home, local and global Lustre filesystems.

The command **ccc_home** displays the personal directories for your login.

```
$ ccc_home -h
ccc_home: Print the path of a user directory (default: home directory).
usage: ccc_home [ -H | -s | -t | -W | -x | -A | -a | -n] [-u user] [-d datadir]
                [-h, --help]
-H, --home          :  (default) print the home directory path ($HOME)
-s, -t, --cccscratch :  print the CCC scratch directory path   ($CCCSCRATCHDIR)
-X, --ccchome       :  print the CCC nfs directory path ($CCCHOMEDIR)
-W, --cccwork       :  print the CCC work directory path  ($CCCWORKDIR)
-A, --cccstore      :  print the CCC store directory path ($CCCSTOREDIR)
-a, --all           :  print all paths
-u user             :  show paths for the specified user instead of the current user
-d datadir          :  show paths for the specified datadir
-n, --no-env        :  do not load user env to report paths
-h, --help          :  display this help and exit
```

By default, members of the same work group have read access to each other's personal spaces.

7.5 Shared Spaces

A shared space is a powerful tool to share environment, installations and results between users or a community. It mainly consists of data storage with additional tools to enhance or manage it. Shared spaces may be used within a community to share:

- Data: input files, configuration files, results, logs

- Products or compute codes

7.5.1 Accessibility and procedures

> **Warning:** Please refer to internal technical documentation to get information about this subject.

7.5.2 Implementation

File systems

Regarding file systems, a shared space is like the already existing personal space: it is spread over 4 different file systems (HOME, CCCSCRATCH, CCCWORK and CCCSTORE). For a shared space called *SHSPACE*, the names to access the 4 different storage spaces will be:

- `$<SHSPACE>_HOME` or `$<SHSPACE>_CCFRHOME`

- `$<SHSPACE>_CCCSCRATCHDIR` or `$<SHSPACE>_CCFRSCRATCH`

- `$<SHSPACE>_CCCWORKDIR` or `$<SHSPACE>_CCFRWORK`

- `$<SHSPACE>_CCCSTOREDIR` or `$<SHSPACE>_CCFRSTORE`

Those environment variables also exist for your personal space: they are named `OWN_HOME`, `OWN_SCRATCHDIR`, etc. and are defined in your default environment.

> **Note:** You may be a member of several shared spaces.
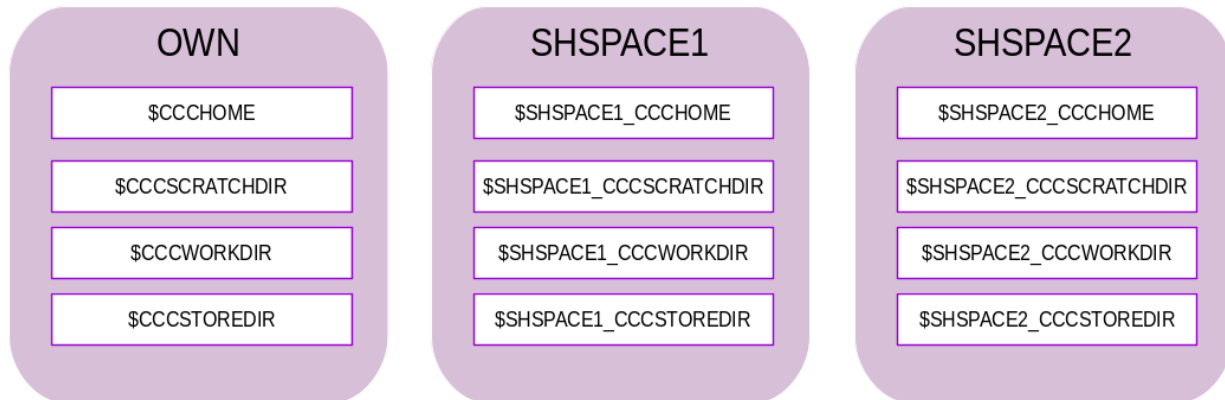

Fig. 1: Shared storage space

Limitations and quota

Several limitations rule a shared space:

- It cannot be shared between users of different containers

- It is not a user account and it does not come with computing hours

Please note that shared file systems have no specific quota. The inodes stored on any file system will impact the quota of their owner.
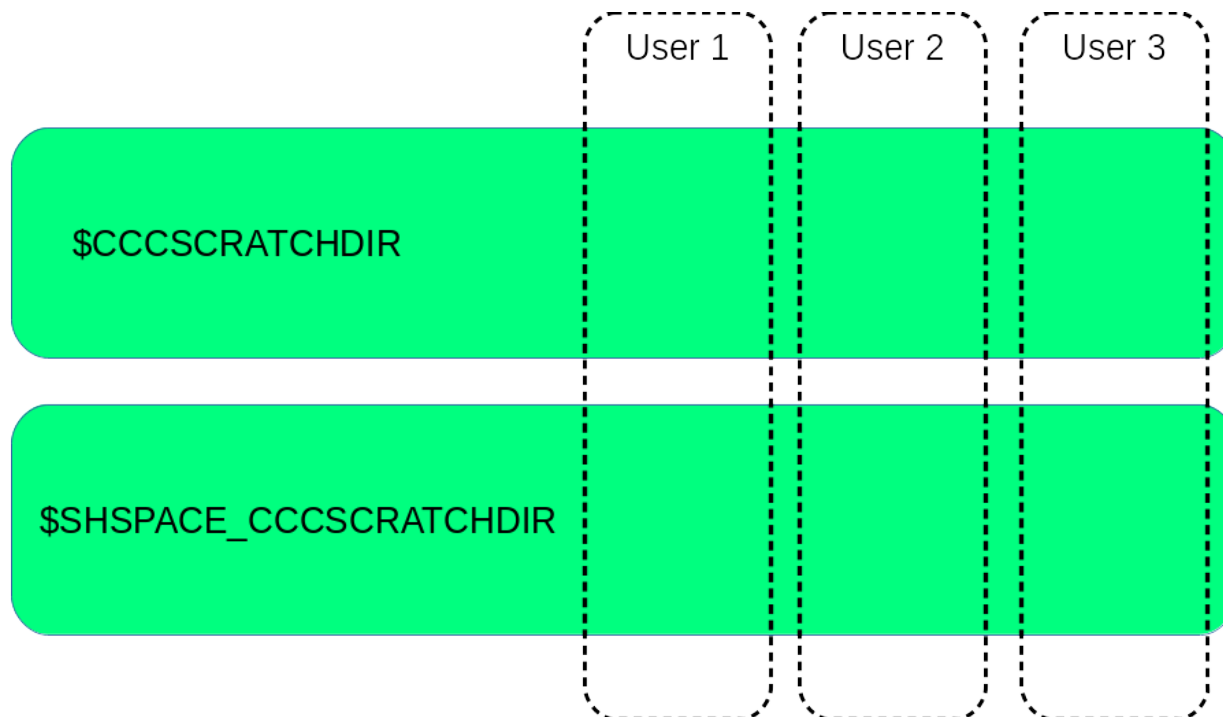
Fig. 2: Shared space quota

Unix groups

> **Warning:** Please refer to internal technical documentation to get information about this subject.

7.5.3 Tools

The computing center provides several tools to ease usage and management of shared spaces.

- The **datadir/<shspace>** and **dfldatadir/<shspace>** modules offer set environment variables that simplify access to the shared storage spaces.

- The **extenv/<shspace>** module helps using shared products or computing codes.

- The scripts **ccc_shspace_chmod** and **ccc_shspace_modck**, along with the module **collwork**, allow to test and set the unix permissions of files and directories in the shared space.

module datadir/dfldatadir

The modules datadir and dfldatadir set environment variables that point to the various storage spaces. Versions of those modules exists for your personal space (own, usually loaded by default), and for each shared space you are member of.

- The datadir module sets an environment variable for each data directory available. Several versions of datadir may be loaded simultaneously in your environment.

The example below shows the default module datadir/own and the impact of loading datadir/shspace:

```
$ module list
Currently Loaded Modulefiles:
1) ccc 2) datadir/own 3) dfldatadir/own

$ echo $OWN_CCCWORKDIR
/ccc/work/contxxx/grp1/user1

$ module load datadir/shspace
load module datadir/shspace (Data Directory)

$ echo $SHSPACE_CCCWORKDIR
/ccc/work/contxxx/shspace/shspace
```

- The dfldatadir module sets the default environment variables for the various storage spaces (CCCSCRATCHDIR, CCCWORKDIR, CCCSTOREDIR and their CCFR equivalent).

Only one version of dfldatadir can be loaded at a time.

This example shows how to use dfldatadir:

```
$ module list
Currently Loaded Modulefiles:
1) ccc 2) datadir/own 3) dfldatadir/own

$ echo $CCCWORKDIR
/ccc/work/contxxx/grp1/user1
```

By default, $CCCWORKDIR (and its CCFR's equivalent $CCFRWORK) points to your own work directory.

```
$ module switch dfldatadir/own dfldatadir/shspace
unload module dfldatadir/own (Default Data Directory)
unload module datadir/own (Data Directory)
load module dfldatadir/shspace (Default Data Directory)
```

Once dfldatadir/shspace is loaded instead of dfldatadir/own, $CCCWORKDIR (and $CCFRWORK) points to your own work directory shared within the shspace.

```
$ echo $CCCWORKDIR
/ccc/work/contxxx/shspace/user1
```

Generic shared variables are set by the dfldatadir module and point to the datadir/shspace variables without mentioning a specific shspace name. This can be useful in order to create generic scripts adapted to different projects.

```
$ echo $ALL_CCCWORKDIR
/ccc/work/contxxx/shspace/shspace
```

Environment extension

The **extenv** module extends the current user environment. It defines a common environment for all users of a shared space.

Loading the extenv module will:

- Set environment variables defining the path to shared products and module files (SHSPACE_PRODUCTSHOME, SHSPACE_MODULEFILES, SHSPACE_MODULESHOME)

- Execute an initialization script

- Add shared modules to the available modules

The environment extension mechanisms uses specific paths. Products installed in the <shspace> shared space should be installed in $SHSPACE_PRODUCTSHOME and the corresponding module files should be in $SHSPACE_MODULEFILES.

Initialization file

If a file named *init* is found in the path defined by $SHSPACE_MODULESHOME, then it is executed each time the extenv/<shspace> module is loaded. This may be helpful to define other common environment variables or to add prerequisites on modules to be used by the community.

For instance the following example defines two environment variables: one is the directory containing input files (SHSPACE_INPUTDIR), and the other is the result directory (SHSPACE_RESULTDIR). It also adds $SHSPACE_PRODUCTSHOME/tools/bin to the PATH so that the tools installed in $SHSPACE_PRODUCTSHOME/tools/bin are easily available.

```
$ cat $SHSPACE_MODULESHOME/init
setenv SHSPACE_INPUTDIR "$env(SHSPACE_CCCWORKDIR)/in"
setenv SHSPACE_RESULTDIR "$env(SHSPACE_CCCSCRATCHDIR)/res"
append-path PATH "$env(SHSPACE_PRODUCTSHOME)/tools/bin"
```

Modules

After the module extenv/<shspace> is loaded, all the module files located in $SHSPACE_MODULEFILES become visible to the module command. For each product, there should be one module file per version.

For example, if you create specific modules in the shared environment in the following paths:

```
$ find $SHSPACE_MODULEFILES
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/code1
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/tool2
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/libprod
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/libprod/1.0
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/libprod/2.0
```

Then, those modules will be visible and accessible once the extenv/<shspace> module is loaded.

```
$ module load extenv/shspace
load module extenv/shspace (Extra Environment)
$ module avail
---------------- /opt/Modules/default/modulefiles/applications ----------------
abinit/x.y.z        gaussian/x.y.z        openfoam/x.y.z
```

(continues on next page)

```
[...]
------------------- /opt/Modules/default/modulefiles/tools -------------------
advisor/x.y.z        ipython/x.y.z          r/x.y.z
[...]
-------- /ccc/contxxx/home/shspace/shspace/products/modules/modulefiles --------
code1         libprod/1.0         libprod/2.0          tool2


$ module load tool2
$ module list
Currently Loaded Modulefiles:
1) ccc           3) dfldatadir/own   5) extenv/shspace
2) datadir/own  4) datadir/shspace  6) tool2
```

Let us consider the example of a product installed in the shared space. The program is called *prog*. Its version is *version*. It depends on the product *dep*. The product should be installed in the directory $SHSPACE_PRODUCTSHOME/prog-version. The syntax of its module file $SHSPACE_MODULEFILES/prog/version would be:

```
#%Module1.0
# Software description
set version      "version"
set whatis       "PROG"
set software     "prog"
set description "<description>"


# Conflict
conflict         "$software"


# Prereq
prereq dep


# load head common functions and behavior
source $env(MODULEFILES)/.scripts/.headcommon
# Loads software's environment


# application-specific variables
set prefix        "$env(SHSPACE_PRODUCTSHOME)/$software-$version"
set libdir        "$prefix/lib"
set incdir        "$prefix/include"
set docdir        "$prefix/doc"


# compilerwrappers-specific variables
set ldflags       "<ldflags>"


append-path PATH "$bindir"
append-path LD_LIBRARY_PATH "$libdir"


setenv VARNAME "VALUE"


# load common functions and behavior
source $env(MODULEFILES)/.scripts/.common
```

- Setting the local variables `prefix`, `libdir`, `incdir` and `docdir` will create the environment variables `PROG_ROOT`, `PROG_LIBDIR`, `PROG_INCDIR` and `PROG_DOCDIR` when the module is loaded. You can also de-

fine any environment variable needed by the software as the example does for `VARNAME`.

- Inter module dependencies are now handled automatically with the *prereq* keyword. That way, any time the module is loaded, its dependencies are loaded if necessary.

We recommend the use of the previous example as a template for all your module files. If you use the recommended path for all installations (`$SHSPACE_PRODUCTSHOME/<software>-<version>`), you can keep that template as it is. Just specify the right software and version and the potential VARNAME environment variables you want to define for the software.

Using that template will ensure that your module behaves the same way as the default modules and that all the available module commands will work for you.

Module collection

The collection mechanism allows to define a set of modules to be loaded either automatically when starting a session, through the *default* collection or manually with `module restore [collection_name]`.

Along with datadir, dfldatadir and extenv, the aim of this collections is to replace all users configurations that may have been done up to now in the shell configuration files. (`.bashrc`, `.bash_profile`, etc)

Here are the useful commands to manage collections:

- To create a collection containing the modules currently loaded in your environment:

```
module save [collection_name]
```

If *[collection_name]* is not specified, it will impact the default collection.

- To list available collections:

```
module savelist
```

- To load a saved collection:

```
module restore [collection_name]
```

If *[collection_name]* is not specified, the default collection will be loaded.

Those collections are stored in the users home, just like shell configuration scripts.

Let us say you have a product Tool2 installed in your shared extended environment. Here is how you would add it to your default collection:

```
$ module list
Currently Loaded Modulefiles:
1) ccc 2) datadir/own 3) dfldatadir/own
```

Load all necessary modules to access your shared product correctly.

```
$ module load datadir/shspace
load module datadir/shspace (Data Directory)
$ module load extenv/shspace
load module extenv/shspace (Extra Environment)
$ module load tool2
load module tool2 (TOOL2)
$ module list
Currently Loaded Modulefiles:
```

(continues on next page)

```
1) ccc        3) dfldatadir/own  5) extenv/shspace
2) datadir/own 4) datadir/shspace 6) tool2
```

Use module save to make the current environment the default one.

```
$ module save
```

After that, every time you will connect to your account, those will be the modules loaded by default.

We highly recommend the use of collections instead of adding calls to **module load** in shell configuration files. A call to **module load** takes non negligible time whether the module was already loaded or not. And when they are specified in ~/.bashrc, they run for each bash script or shell execution. On the contrary, a collection is only loaded if the environment has not yet been initialized. Therefore, using collections will fasten connections and script executions.

Note: To ensure the correct setting of your user environment, make sure

- That the file ~/.bashrc contains:

```
if [ -f /etc/bashrc ]; then
. /etc/bashrc
fi
```

- That the file ~/.bash_profile contains:

```
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi
```

Managing permissions

Warning: Please refer to internal technical documentation to get information about this subject.

7.6 Parallel file system usage monitoring

Some general best practice rules should be followed when using the available file systems. They are specified in *Recommended data usage on parallel file system*.

Inappropriate usage of parallel file systems is monitored, as it may badly affect overall performances. Several inappropriate practices are tracked down.

Warning: Non-compliance with the appropriate file system usage guidelines may trigger an alert, requiring immediate user attention and action.

7.6.1 Too many small files on CCCSTOREDIR

Users with more than 500 files on STORE are under monitoring and will be issued E-mail alerts if they have:

- an average file size below 1GB (see the column "avg_fsize" from `ccc_quota`),

- or the percentage of files smaller than 500MB is higher than 80% (see the column "files<500M" from `ccc_quota`).

If the STORE rules are not followed, the following measures will be taken until the situation returns to normal:

- Day D: The user is informed by e-mail.

- Day D+14: The user is informed by e-mail and their submissions are locked.

- Day D+28: The user is informed by e-mail, their account is locked and their files may be archived or even deleted.

7.6.2 Too many files in a directory on SCRATCH, WORK and STORE

Having a directory with more than 50000 entries (files or directories) at the root level is inappropriate.

Once a week if this rule is not followed, actions are triggered until situation is improved:

- D0: The user is informed by mail.

- D7: The user is informed by mail.

- D14: The user is informed by mail and his submissions are locked.

- D21: The user is informed by mail.

- D28: The user is informed by mail, his account is locked and his files can be removed.

7.7 Data management on STORE

7.7.1 ccc_pack

ccc_pack allows for a storing large volumes of data. Usually, it is used to transfer data from the Scratch to the Store. Here is an example case:

```
$ ccc_pack --mkdir --partition=<partition> --filesystem=store,scratch --src=
→$CCCSCRATCHDIR/<source file path>   --dst=$CCCSTOREDIR/<destination directory> -j 2
$ ccc_msub $CCCSTOREDIR/<destination directory>/pack_<ID>.msub
```

This command is a program to pack sources files or directory according to destination filesystem properties. Current filesystem properties are minimum, optimum, and maximum filesize.

The program will chroot content path in toc and pack to source path. If multiple sources are supplied, files with the same name and relative path will be packed. After packing no information will differentiate them (for extraction).

Symlinks will not be followed by the pack, they will be archived as a symbolinc link. It write a block to the pack naming the target of the link.

During program execution, no source file is locked. If the file is modified, we can not guarantee which version is saved. At the end of execution, files are not deleted. If necessary these actions must be performed by the user.

By default this tool will produce:

- For file less than maximum filesize pack: x packs named basename-part.tar

- For file greater than maximum filesize pack: x packs named basename-part-volume.split

- For each pack: a table of contents basename-part.toc, a file with md5sum of each table elements basename-part.md5, a file with md5sum of each pack basename.md5

If Scheduling is activated, this tool will produce:

- A file with tasks to run in parallel: MOD_PREFIXbasename.glost

- A file to submit: MOD_PREFIXbasename.msub

Note: More options are available, please use **`ccc_pack --help`** or **`man ccc_pack`** for more information.

7.7.2 ccc_unpack

ccc_unpack is the routine to reverse a pack. Usually, it is used to transfer data back from the Store to the Scratch. Here is an example case :

```
$ ccc_unpack --src-basename=ID --src=$CCCSTOREDIR/<archive directory> --dst=
↪$CCCSCRATCHDIR/<output dir> --partition=milan -j 2

$ ccc_msub $CCCSTOREDIR/unpack_<ID>.msub
```

More options are available, please use **`ccc_unpack --help`** or **`man ccc_unpack`** for more information.

7.7.3 ccc_hsm

`ccc_hsm` is a tool designed to manage data among multiple levels of storage devices, from Lustre SDDs/HDDs to an archival system such as HPSS or Phobos. **`ccc_hsm`** can manage data placement through hints by migrating data within Lustre storage components. It can also trigger the copy of archived data back to Lustre through Lustre-HSM mecanisms. The details of its use is described in the following paragraphs.

ccc_hsm on the STORE

Display status

```
$ ccc_hsm status <file>
```

You can use the `-T` option to also display the storage tiers if the file is online, or its archive id if released.

Here is an example of an output on the store

```
$ ccc_hsm status -T test_dir/test_file
test_dir/test_file                    online  (hdd)
```

In this case, the file is ready to be used.

```
ccc_hsm status -T test_dir/test_file2
test_dir/test_file2      released
```

In this case, the file need to be fetched

To show the HSM status of all files in a directory, use the command:

```
ccc_hsm ls <dir>
```

In the same way, you can add the `-T` option to display storage tiers.

Here is an example of an output:

```
ccc_hsm ls -T test_dir
online   (hdd)          test_dir/test_file_1
online   (flash)        test_dir/test_file_2
released (1)            test_dir/test_file_3
```

Retrieving data

To retrieve files, you can use the **get** command to preload the entries into Lustre. The given path to retrieve can also be a directory, in which case additional options are required to specify the behaviour. The command is:

```
ccc_hsm get <path>
```

Multiple options can be added:

- `-r`: for recursive get if the given path is a directory

- `-b`: to get the data asynchronously in background

- `-d`: to only retrieve the first level entries of a directory

- `-n`: to only try to preload files once

If the retrieval is blocking (i.e. not using the `-b` option), it will wait for the file to be online. However, if the file isn't online by the time the timeout is reached, **ccc_hsm** has no way of knowing why it couldn't be retrieved.

Advanced usage for the store

ccc_hsm allows one to set special hints on a file to indicate the usage of the file in the future. This will help optimize the system performance and efficiently choose how and when migrations and archival are done.

> **Warning:** Writing rights are required to use hints on a file. You may use hints only on the STORE and SCRATCH filesystems.

This is done by using the command:

```
ccc_hsm hint <hint> <path>
```

Multiple hints can be declared in the command line, and they have the following meanings:

- `--wont-access`: The specified data will not be accessed in the near future (incompatible with `--will-access`)

- `--wont-change`: The specified data will not be modified in the near future

- `--no-archive`: The specified data should not be archived in the near future

- `--will-access`: The specified data will be accessed in the near near future (incompatible with `--wont-access`)

- `--clear`: Remove all previous user hints before applying new ones (if provided)

If no hint is provided, it will display the hint currently attached to the file.

For instance, if you know the file will not be accessed in the near futur, you may use the following command :

```
$ ccc_hsm hint --wont-access test_dir/test_file1
$ ccc_hsm hint test_dir/test_file1
wont-access
```

7.8 Data management on SCRATCH

7.8.1 ccc_will_purge

ccc_will_purge displays all files to be purged within 15 days. You may use the `-d` option to change the delay :

```
ccc_will_purge -d <maximum number of days before the purge>
```

7.8.2 ccc_hsm

`ccc_hsm` is a tool designed to manage data among multiple levels of storage devices, from Lustre SDDs/HDDs to an archival system such as HPSS or Phobos.

> **Warning:** The SCRATCH is not a HSM Filesystem

ccc_hsm on the SCRATCH

Display status

```
ccc_hsm status <file>
```

You can use the `-T` option to also display the storage tiers if the file is online, or its archive id if released.

Here is an example of an output on the scratch:

```
$ ccc_hsm status -T test_dir/test_file
test_dir/test_file              online  (hdd)
```

In this case, the file is on the hard drive (hdd). This is recommended if :

- the files are not accessed often, or more generally if IO performances are not relevant
- the access order is consistent

This pool is also less used and may provide better writing performances when the machine load is heavy.

```
$ ccc_hsm status -T test_dir/test_file
test_dir/test_file              online  (flash)
```

In this case, the file is on the NVME (flash memory). This is recommended if :

- The files are accessed often, in a random order
- The IO performances are crucial to the code

`ccc_hsm` allows one to set special hints on a file to indicate the usage of the file in the future. This will help optimize the system performance and efficiently choose how and when migrations and archival are done.

> **Warning:** Writing rights are required to use hints on a file. You may use hints only on the STORE and SCRATCH filesystems.

This is done by using the command:

```
ccc_hsm hint <hint> <path>
```

Multiple hints can be declared in the command line, and they have the following meanings:

- `--wont-access`: The specified data will not be accessed in the near future (incompatible with `--will-access`)

- `--wont-change`: The specified data will not be modified in the near future

- `--no-archive`: The specified data should not be archived in the near future

- `--will-access`: The specified data will be accessed in the near near future (incompatible with `--wont-access`)

- `--clear`: Remove all previous user hints before applying new ones (if provided)

If no hint is provided, it will display the hint currently attached to the file.

For instance, if you know the file will not be accessed in the near futur, you may use the following command :

```
$ ccc_hsm hint --wont-access test_dir/test_file1
$ ccc_hsm hint test_dir/test_file1
wont-access
```

7.9 Large files management

For files larger than 10 Go on the STORE, WORK or SCRATCH, it is recommanded to "strip" the file: Stripping a file consist in dividing it between several Object Storage Targets (OSTs).

In order to strip an existing file you may use the following commands:

```
lfs setstripe -c <number of stripes> <new path to file>
cp <previous path> <new path to file>
```

In this command, the `-c` option specifies the stripe count, which is the number of OSTs that the file will be striped across. In most cases, 2 to 4 stripes are enough to handle large files. In cas your IO patterns are well parallised, with the least amount of conflicts, you may use more stripes. Nevertheless, you may use no more than one OST per 32GB of file and no more than 16 stripes. Adding striping will increase read/write bandwidth and decrease latency fo large files but will increase the number of requests generated and thus the load on the processor.

For faster striping of the file, you may use the MPIFileUtils command dstripe as described in the *MPIFileUtils subsection*, in the Paralell IO section.

Once you have run this command, Lustre will automatically distribute the file across the specified number of OSTs. You can confirm that the file is striped across multiple OSTs by checking the output of the **lfs getstripe** command:

```
lfs getstripe <new path to file>
```

This will show you the stripe count, stripe size, and OSTs that the file is striped across.

You may also define the stripping for a directory, which will affect all new files created in it:

```
lfs setstripe -c <number of stripes><path of directory>
```

By default, maximum file size is 4TB, stripping allows for bigger files. If needed, you may use the command:

```
ulimit -f <file size limit>
```

However its usage is not recommended. In case of need, please send an email to the hotline.

In order to have the best IO performance, you may directly directly stripe your outputs with *MPI IO*. More details can be found in the I/O section of the official MPI documentation.

7.10 Cloud storage with S3 (MinIO)

7.10.1 Presentation

MinIO is a S3 High Performance Object Storage for machine learning, analytics and application data workloads. It allows you to easily store and access large amounts of data with a user-friendly API. It is designed to handle a large volume of data, and ensure that it is always available, reliable and secure. MinIO is ideal for storing any type of unstructured data that can continue to grow.

7.10.2 S3 data spaces

Two kinds of S3 data spaces can be associated with your project:

- A TGCC S3 data space, by default the project name is used, i.e. <shspace>. Only TGCC users of your project can read or write in this S3 data space.

- A PUBLIC S3 data space, by default the project name is used and suffixed "-pub", i.e. <shspace>-pub. Objects stored in this S3 data space are available from the Internet without authentication. For more details, please refer to the "Public sharing" section.

7.10.3 Prerequisite

In order to use MinIO, you have to be part of an European or GENCI project.

7.10.4 MinIO project management

Project management at the TGCC

First, you can display your S3 project:

```
$ module avail s3
----------------------- /ccc/etc/modulefiles/environment -----------------------
s3/info  s3/<shspace>  s3/<shspace>-pub
```

In order to use MinIO, you need to load your S3 project module:

```
$ ml s3
This module can not be load!
You can display more informations with 'module help s3/info' command.

$ ml help s3/info
-------------------------------------------------------------------
Module Specific Help for /ccc/etc/modulefiles/environment/s3/info:

##### S3 MODULES INFO #####
```

(continues on next page)

```
1. List your s3 modules with 'module avail s3':
 s3/<shspace>  s3/<shspace>-pub

2. Load your project associated module with 'module load s3/<shspace>'

3. Use 'mc' command with $CCCS3DIR or $<shspace>_CCCS3DIR:
 $ mc ls $CCCS3DIR

For any question, please contact your hotline.
------------------------------------------------------------------

$ ml s3/<shspace>
load module miniocli/2022.12.02
===
MINIO_ACCESS_KEY_ID, MINIO_SECRET_ACCESS_KEY and MINIO_SESSION_TOKEN variables are
↪available!
===
load module minio/2024.05.09
```

Variables MINIO_ACCESS_KEY_ID, MINIO_SECRET_ACCESS_KEY and MINIO_SESSION_TOKEN are available. You need theses variables to set-up your minio-py API client.

Project management from an authorized connection point

Get the following files script provided by TGCC:

```
cea_s3_token_[eu-fr].sh
```

For GENCI projects:

```
$ curl  "https://data-pub-fr.ccc.cea.fr/?container=public_resources&object=cea_s3_token_
↪fr.sh" --output cea_s3_token_fr.sh
$ source cea_s3_token_fr.sh token
```

For European projects:

```
$ curl  "https://data-pub-eu.ccc.cea.fr/?container=public_resources&object=cea_s3_token_
↪eu.sh" --output cea_s3_token_eu.sh
$ source cea_s3_token_eu.sh token
```

7.10.5 S3 usage

Informations and statistics

You can display various informations of your account, container or objects with the mc command:

```
$ mc ls $CCCS3DIR/
$ mc stat $CCCS3DIR/bucket1
```

Uploads and dowloads

Upload files and directories to a given container.

```
$ mc mb $CCCS3DIR/mynewbucket
$ mc cp source_file1.txt $CCCS3DIR/mynewbucket/
```

Download all files in a container

```
$ mc cp --recursive $CCCS3DIR/mynewbucket/ ./
```

Download a file in a container

```
$ mc cp $CCCS3DIR/mynewbucket/source_file1.txt ./
```

Suppression

Delete files in a container

```
$ mc rm $CCCS3DIR/mynewbucket/source_file1.txt
```

Delete a file in a container

```
$ mc rb $CCCS3DIR/mynewbucket
```

7.10.6 Sharing your project

Authentified sharing

In order to share your files with people in your community, you need to send a mail to the hotline.

Public sharing

If you want to make your files public, you may also make a request to the hotline. Then anyone can access your files without authentication in read-only with the following command:

For GENCI projects:

Read Object

```
$ curl  "https://data-pub-fr.ccc.cea.fr/get?container=test1&object=test2/picture.png" --
↪output picture.png
```

List bucket

```
$ curl  "https://data-pub-fr.ccc.cea.fr/list?container=test1
```

For European projects:

Read Object

```
$ curl  "https://data-pub-eu.ccc.cea.fr/get?container=test1&object=test2/picture.png" --
↪output picture.png
```

List bucket

```
$ curl "https://data-pub-eu.ccc.cea.fr/list?container=test1"
```

DATA TRANSFERS

8.1 File transfer

The center provides several means of transferring data to and from its various resources.

- On Unix-like OSes (for instance Linux or Mac OS X) use the **scp** or **rsync** commands.

- On Windows, several clients exist. For example **PSCP** or **WinSCP**

Note: SFTP (SSH File Transfer Protocol), which is used for example by **WinSCP**, is disabled for security reasons on European and FR login nodes.

Call the transfer routines from your local machine (*local_host*). Here are the different ways to copy data from or to the supercomputer login node (*remote_host*). The remote host depends on the type of project. It is the one used for SSH connections (see *Interactive access* for more details). Hereafter, remote_dir represents a valid directory on the remote host.

Note:

- If you search [...] you can look at the Data spaces chapter, at the Data management section, and at the Parallel IO chapter, at the MpiFileUtils sections.

8.1.1 scp

scp copies files between hosts on a network. It uses ssh for data transfer, with the same authentication and the same security as **ssh**.

To transfer data from local machine to remote machine:

```
$ scp [options] <local_files> <login>@<remote_host>:<remote_dir>
```

To transfer data from remote machine to local machine:

```
$ scp [options] <login>@<remote_host>:<remote_dir>/<files> <local_dir>
```

Basic options are -v for verbose mode and -r to copy directories. For more information, type **man scp** from the command line.

8.1.2 rsync

rsync synchronizes two sets of files across a network. It sends only the differences between the source files and the destination files.

To transfer data from local machine to remote machine:

```
$ rsync -e ssh -avz <local_files> <login>@<remote_host>:<remote_dir>
```

To transfer data from remote machine to local machine:

```
$ rsync -e ssh -avz <login>@<remote_host>:<remote_dir>/<files> <local_dir>
```

For more information, type **man rsync** from the command line.

Note: Transferring data from or to the supercomputer may be more efficient when using archives instead of many small files.

8.1.3 sftp

sftp connects on a remote host, then can transfer files on both directions. It uses ssh for data transfer, with the same authentication and the same security as ssh.

To connect on a remote host:

```
$ sftp [options] <login>@<remote_host>
```

An usual option is `-r` to copy directories. For more information, type man sftp from the command line.

Once logged in, to transfer data from the remote host to the local host:

```
$ sftp> get [options] <remote_path> <local_path>
```

And to transfer data from the local host to the remote host:

```
$ sftp> put [options] <local_path> <remote_path>
```

If you haven't used the `-r` option with the **sftp** command, you can use it directly with **get** and **put**. For more information, type **help** from the **sftp** prompt, or **man sftp** from the command line.

8.1.4 parallel sftp

To speed up file transfer, you can use **parallel-sftp**, a tool developed by CEA. **parallel-sftp** uses parallel **sftp** to make file transfers. You can install it locally by following the dedicated website: https://github.com/cea-hpc/openssh-portable

---

Note:

- parallel-sftp is not tested on Windows and MacOS.

- parallel-sftp is only useful when your transfer overuses a CPU on one of the hosts (local or remote computer). On a modern processor, it happens when the transfer bandwidth reaches $\pm$ 1Gbps (please double check if the network link between both nodes can use more than 1Gbps of bandwitdh).

---

parallel-sftp is installed on login nodes of CEA clusters. Its usage is identical as **sftp**, except the option -n which let you choose the number of ssh connections used for the parallel transfer.

For example, to make a parallel transfer with 5 ssh connections:

```
$ parallel-sftp -n 5 <login>@<remote_host>
```

If one **sftp** transfer is limited at 1Gbps, this transfer will use at most 5Gbps.

Note: If you search for handling files within the cluster, you can look at the *Parallel IO* chapter and the *MpiFileUtils* section.

8.2 CCFR infrastructure

Warning: Please refer to internal technical documentation to get information about this subject.

8.3 PRACE infrastructure

Warning: Please refer to internal technical documentation to get information about this subject.

NINE

ENVIRONMENT MANAGEMENT

Traditional Unix way to manage environment usually involves editing your `~/.bashrc` and/or sourcing software-specific files. This methodology can be error-prone due to inconsistent definitions and hardly let users dynamically enable or change the software they want to use. To dynamically manage your environment and pick up the needed software among the large software catalog provided, the **module** command is provided from the Environment Modules project.

9.1 What is module

`module` is a user interface providing dynamic modification of your environment via modulefiles. `module` allows to change easily the shell environment by initializing, modifying or unsetting environment variables.

Each modulefile contains the information needed to configure the shell for an application. Once the module is initialized, the environment can be modified on a per-module basis using the module command which interprets modulefiles. Typically modulefiles instruct the module command to alter or set shell environment variables such as `PATH`, `MANPATH`, etc. The modulefiles are added to and removed from the current environment by the user.

On the computing center, it is typically used to:

- define your user environment (`$HOME`, `$CCCSCRATCHDIR`, etc.);
- easily get access to third-party softwares in different versions (ex: Intel compilers, GCC, MPI librairies, etc.);
- handle the potential conflicts an requirements between software.

Modulefiles are basically provided by the computing center staff to get access to the installed software and to the system properties. In addition you may have your own modulefiles to supplement the already provided modulefiles.

9.2 Module actions

Major kind of actions of the module command are described below. To get a full reference of the available module actions, you can either

- display the command usage message (`module -h`)
- look at the man page (`man module`)
- or even try the command auto-completion

9.2.1 Listing / Searching modulefiles

Knowing current environment state:

- **module list** shows the current state of your environment, which means to display all the modules currently loaded

Querying modulefiles catalog:

- **module avail** displays all available modules (modules suffixed by @ are aliases)
- **module avail --default** reduces the regular *avail* output by only displaying the available default versions
- **module avail --latest** in the same way displays only the available latest versions
- **module avail mpi** shows the available *mpi* modulefiles

Printing modulefile information:

- **module help netcdf** shows software description for default *netcdf* modulefile
- **module help hdf5/x.y.z** shows software description for a specific *hdf5* modulefile
- **module show mkl** displays software description plus environment definition for the default *mkl*
- **module show python/x.y.z** displays as above description and environment definition but for a specific version

Searching for modulefiles:

- **module whatis papi** prints for each version of *papi* product a one-liner description and its associated keywords
- **module help|show products/keywords** prints *products/keywords* modulefile description which lists all keywords in-use by available modulefiles
- **module search profiler** searches for all products whose name, one-liner description or keywords match the *profiler* search string

9.2.2 Loading / Unloading modulefiles

Adding modulefile(s) to the list of currently loaded modules:

- **module load fftw3** loads the default version of *fftw3* product or its latest version if no default version is explicitly set
- **module load visit/x.y.z** loads specific version *x.y.z* of *visit*
- **module load intel hdf5 netcdf** loads multiple modulefiles in one command

---

Note: On interactive shells, module auto-completion is enabled and can help you to find the name of modulefiles you want to load, unload or switch

---

Removing modulefile(s) from your current environment:

- **module unload visit** unloads loaded version of *visit* modulefile
- **module unload netcdf hdf5** unloads multiple products in one command
- **module purge** unloads all loaded modulefiles

Note: All these load, unload, switch commands returns 0 on success or 1 elsewhere

---

Switching from one version of a modulefile to another:

- **module switch intel intel/x.y.z** unloads currently loaded *intel* modulefile then loads version *x.y.z*

Note: The **module** command will automatically satisfy modulefile prerequisites. When loading a modulefile, all the modulefiles it declares as prerequisite are loaded prior to its own load. When unloading a modulefile, all the modulefiles it declares as prerequisite that have been automatically loaded as dependency are automatically unloaded after the initial modulefile unload.

9.2.3 Saving and restoring modulefile collections

A modulefile collection corresponds to saved state of your module environment you can restore whenever you want. A collection is composed of an ordered set of modulefiles which are the currently loaded modulefiles at the time of saving this collection. When a collection is restored, currently loaded modulefiles are unloaded to then load the set of modulefiles defined in the collection in the same loading order.

You can own any number of collections you want, which gives you the ability to easily switch between a production environment and a development environment or between a visualization environment and a debugging one, for instance.

Saving modulefile collections

- **module save development** saves the current list of loaded modules in the collection named *development*
- **module save** saves the current list of loaded modules in the *default* collection

Listing saved modulefile collections

- **module savelist** lists all previously saved modulefile collections

Restoring saved modulefile collections:

- **module restore development** restores the collection named *development*, by unloading currently loaded modulefiles then loading the modulefiles defined in the collection to restore the same ordered list of loaded modulefiles
- **module restore** restores the *default* modulefile collection

9.3 Initialization and scope

Depending on shell mode, the **module** environment is initialized and propagated in different ways. In all cases, the **module** command is defined and a minimal environment is set. This minimal environment is composed of the default paths to the modulefiles provided by the computing center staff and the mandatory modulefiles *ccc* and *dfldatadir* loaded. Then on interactive shell:

1. all **module** output messages are set to be redirected to stdout
2. **module** command auto-completion is enabled
3. your module collection named *default* is restored if it exists

On non-interactive shell following initialization is done after minimal environment setup:

1. all **module** output messages are let on stderr
2. module message at load or unload is disabled
3. your module collection named *non-interactive* is restored if it exists

---

9.3.1 Interactive or non-interactive ?

Interactive shell initialization is obtained when:

- you connect to the supercomputer or to a given node within the supercomputer to get a login shell
- you run an interactive job

Non-interactive shell initialization is obtained elsewhere, which means when:

- a batch job starts
- you remotely execute a command (with SSH) on the supercomputer or on a node within the supercomputer

9.3.2 Scope of your environment

Your environment is initialized or re-initialized in the conditions previously described, which means each time you connect to the supercomputer or from one node to another within the supercomputer your environment is reset to its default. It is also the case when the batch scheduler starts one of your batch job: by default the environment at the time of the job submission is not restored so the environment of this batch job is initialized as a non-interactive environment.

Once initialized, each load or unload of modulefile modifies the environment of the current shell, its subsequent sub-shells and jobs. So each sub-shell and script launched will inherit the environment from its parent shell. However to guaranty the **module** function to still be defined in sub-shells and script launched, please ensure that `/etc/bashrc` configuration is loaded in your `~/.bashrc` local configuration file, for instance with:

```
# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
```

Regarding interactive jobs, the environment obtained at the start of this kind of job is the one set when you call for these interactive jobs. Which means all the modulefiles loaded prior to the execution of an interactive job will be still loaded once this interactive job session will be established.

9.4 Major modulefiles

Modulefiles provided by the computing center staff are spread across categories to sort them by product function. These categories are *applications*, *compilers*, *environment*, *libraries*, *tools*, *parallel* and *graphics*. They represent the type of software, except for *parallel* and *graphics* categories which are transversal to the other categories and are made to promote these kind of software.

The *environment* category is a bit special as it does not contain modulefile definition for product like the other categories. A modulefile from the environment category defines aspects relative to the system or configuration properties for a group of user that can access and use these system functions.

Some modulefiles from the *environment* category shape environment usages and possibilities. These major modulefiles are described below.

9.4.1 ccc

`ccc` modulefile defines global system variables and aliases needed to get a functional user environment. This modulefile is the first to be loaded in user environment and its load is done within *default environment load*. It is mandatory and thus cannot be unloaded.

9.4.2 datadir

`datadir` modulefile helps user to localize his own or shared/application space-specific data end points. Personal or shared spaces provide multiple data end-points each for different needs. These endpoints are localized with environment variables to easily reference these various end-points. Data end-points variables are all prefixed with the name of the module version upper-cased, which corresponds to the name of the personal or shared space. For instance, data end-points for project `prj` will be all prefixed by `PRJ_` and end-points for personal space will be all prefixed by `OWN_`.

A version for the `datadir` modulefile exists for each shared/application space known in the computing center, named in accordance to the shared space name, and for the user personal space, named `own`. You can only view and access versions of the `datadir` modulefile that correspond to spaces you can access. Multiple versions of the `datadir` module can be loaded at the same time.

Following example display the variables set for the own version of the `datadir` modulefile:

```
$ module show datadir/own
-------------------------------------------------------------------
/opt/Modules/default/modulefiles/environment/datadir/own:

setenv        OWN_STOREDIR    /ccc/store/contxxx/group/username
setenv        OWN_CCCSTOREDIR /ccc/store/contxxx/group/username
setenv        OWN_WORKDIR /ccc/work/contxxx/group/username
setenv        OWN_CCCWORKDIR  /ccc/work/contxxx/group/username
setenv        OWN_SCRATCHDIR  /ccc/scratch/contxxx/group/username
setenv        OWN_HOME    /ccc/contxxx/home/group/username
module-whatis   Data Directory
-------------------------------------------------------------------
```

9.4.3 dfldatadir

`dfldatadir` modulefile sets personal or shared/application space data end-points targeted by a *datadir* modulefile as default end-points. Data end-point variables set by `datadir` modulefile are all prefixed with the name of the module version upper-cased, `dfldatadir` module sets the default data end-point variable (without prefix) for each of these personal or shared-space specific variables set by corresponding `datadir` modulefile. Exception is made for `HOME` variable which always refer to personal home directory and do not change if a `dfldatadir` modulefile different than default is loaded.

A version for `dfldatadir` modulefile exists for each shared/application space known in the computing center, named in accordance to the shared space name, and for the user personal space, named `own`. You can only view and access versions of the `datadir` modulefile that correspond to spaces you can access. Since `dfldatadir` module represents default data end-points, only one version of the module can be loaded at the same time.

`dfldatadir` modulefile requires the *datadir* modulefile with same version name. This `datadir` modulefile is thus automatically loaded when loading the `dfldatadir` modulefile. Default version of the `dfldatadir` module is the `own` modulefile, which is loaded by default when module environment is initialized.

To guaranty a coherent user environment with default datadir locations set (`CCCSCRATCHDIR`, `CCCSTOREDIR`, etc) `dfldatadir` is mandatory and thus cannot be unloaded. As a consequence, to change loaded `dfldatadir` module **module switch** command has to be used as **module unload** will fail:

```
$ module unload dfldatadir
module dfldatadir/own (Default Data Directory) cannot be unloaded
$ module switch dfldatadir/own dfldatadir/prj
unload module dfldatadir/own (Default Data Directory)
unload module datadir/own (Data Directory)
load module datadir/prj (Data Directory)
load module dfldatadir/prj (Default Data Directory)
```

9.4.4 extenv

`extenv` modulefile enables users to extend the environment provided by the computing center staff with extra environment managed within its own home directory or within a shared/application space home directory. `extenv` introduces a standard layout to manage your software products through the module environment provided by the computing center. Details on this modulefile are exposed in *the next section*.

9.4.5 feature

`feature` modulefile enables users to adjust the settings of a product through environment variables. `feature` introduces a standard layout to manage your software products settings through the module environment provided by the computing center. Typical usages are:

- **module whatis feature** to list products with multiple settings
- **module whatis feature/openmpi** to list and describe the available settings for the software *openmpi*

`feature` adjust the behaviour of a product without software recompilation. When behaviour changes requires dedicated build it will be in the *flavor* modulefiles.

The following example illustrates how the `feature` modules for MKL load some environment variables that affect the product in some way of its own. You can list those features with **module av feature/mkl**. Here is what they do:

```
$ module load feature/mkl/sequential
$ module load mkl
$ module show mkl
...
module-whatis Intel MKL LP64 Sequential
setenv       MKL_LDFLAGS ... -lmkl_sequential ...   # Here, link options will change
...
$ module switch feature/mkl/{sequential,multi-threaded}
$ module show mkl
...
module-whatis Intel MKL LP64 Multi-threaded
setenv       MKL_LDFLAGS ... -lmkl_intel_thread ... #... here we see the change
...
```

9.4.6 flavor

`flavor` modulefile enables users to select a specific build for a product. `flavor` introduces a standard layout to manage the multiple compilations for a same software products through the module environment provided by the computing center. Typical usages are:

- **module whatis flavor** to list the products with multiple compilations/builds

- **module whatis flavor/hdf5** to list and describe the available compilations/builds for the software *hdf5*

`flavor` adjust the behaviour of a product with dedicated software compilation. When behaviour changes does not requires it, it will be in the *feature* modulefiles.

Here is a more detailed illustration of what `flavor` modules do:

- `flavor/%product%/%wish%` expresses a wish about the installed version of a product you want to choose, but it does not load anything by itself;

- `%product%/%version%` points to the installation of the product, depending on the `flavor` modules you may have loaded previously;

Here is a real-life example: **module av flavor/hdf5** mentions `parallel` and `serial`. Let's try both:

```
$ module load flavor/hdf5/serial
$ module load hdf5
$ module show hdf5
...
prepend-path  PATH   /.../serial/bin # Installation paths change with the flavor
...
$ module load mpi  #HDF5 parallel will require a MPI implementation
$ module switch flavor/hdf5/serial flavor/hdf5/parallel
$ module show hdf5
...
prepend-path  PATH   /.../parallel/bin # a new flavor changes the installation path
```

9.4.7 licsrv

`licsrv` modulefile defines in user environment the variables required by software to query the license server they are related to. Each version of this modulefile represents an existing license server. `licsrv` modulefile is automatically loaded when loading a modulefile who requires the relative license server.

9.4.8 products

`products` modulefiles provides functions to query the product catalog. These modulefiles can only be displayed, they are not intended to be loaded. They provide different kind of information on the installed software.

- **module help|show products/keywords** displays all the existing product keywords

- **module help|show products/newinstall** lists all the software versions whose installation date is fresher than 8 weeks

- **module help|show products/restrict** lists all the software whose usage is restricted and your current grant status for these software.

9.5 Extend your environment with modulefiles

Computing center staff provides you regular HPC software you can access through the `module` environment. You may need to extend this regular environment with your own product installations or various setups. This section describes how to enable your environment extensions within the `module` environment.

9.5.1 Using the `extenv` modulefile

`extenv` modulefile enables users to extend the environment provided by the computing center staff with extra environment managed within its own home directory or within a shared/application space home directory. `extenv` introduces a standard layout to manage your software products through the module environment provided by the computing center. This modulefile enables to define a common environment for all users of a given shared space.

A version for `extenv` modulefile exists for each shared space known in the computing center, named in accordance to the shared space name, and for the user personal space, named `own`. You can only view and access versions of the `extenv` modulefile that correspond to spaces you can access. Multiple versions of the `extenv` module can be loaded at the same time.

Loading the `extenv` modulefile will:

- Set environment variables defining the path to shared products and modulefiles (SHSPACE_PRODUCTSHOME, SHSPACE_MODULEFILES, SHSPACE_MODULESHOME)
- Execute a module initialization script
- Add shared modulefiles to the list of available modulefiles

The environment extension mechanisms of `extenv` requires the use of specific paths. Products installed for the shared space named *shspace* should be installed in $SHSPACE_PRODUCTSHOME and the corresponding modulefiles should be in $SHSPACE_MODULEFILES.

Initialization file

If a file named `init` is found in the path defined by $SHSPACE_MODULESHOME, then each time the `extenv/shspace` module is loaded, this initialization file will be executed as TCL code. This may be useful if you want to define other common environment variables or add prerequisites on modules to be used by the community.

For instance, with the following example, you will define two environment variables, one defining the path to a directory containing input files (SHSPACE_INPUTDIR), and the other defining the result directory (SHSPACE_RESULTDIR). It will also add $SHSPACE_PRODUCTSHOME/tools/bin to the PATH so that the tools installed in this directory are easily available.

```
setenv SHSPACE_INPUTDIR "$env(SHSPACE_CCCWORKDIR)/in"
setenv SHSPACE_RESULTDIR "$env(SHSPACE_CCCSCRATCHDIR)/res"
append-path PATH "$env(SHSPACE_PRODUCTSHOME)/tools/bin"
```

Expose modulefiles

Once the `extenv/shspace` modulefile is loaded, all the modulefiles located in `$SHSPACE_MODULEFILES` will be visible to the **module** command. For each product, there should be one module file per version. You can also define modulefiles for configuration or environment change, it is not mandatory to relate each modulefiles to a product.

For example, if you create specific modules in the shared environment in the following paths:

```
$ find $SHSPACE_MODULEFILES
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/thecode
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/thecode/1.2.3
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/thetool
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/thetool/2
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/conf
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/conf/thecode
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/conf/thecode/production
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/conf/thecode/tuningtest
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/libprod
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/libprod/1.0
/ccc/contxxx/home/shspace/shspace/products/modules/modulefiles/libprod/2.0
```

Then, those modules will be visible and accessible once the extenv/<shspace> module is loaded.

```
$ module load extenv/shspace
load module extenv/shspace (Extra Environment)
$ module avail
---------------- /opt/Modules/default/modulefiles/applications -----------------
abinit/x.y.z        gaussian/x.y.z        openfoam/x.y.z
[...]
------------------- /opt/Modules/default/modulefiles/tools -------------------
advisor/x.y.z        ipython/x.y.z        r/x.y.z
[...]
-------- /ccc/contxxx/home/shspace/shspace/products/modules/modulefiles --------
thecode        conf/thecode/production    conf/thecode/tuningtest
libprod/1.0    libprod/2.0                thetool/2


$ module load thetool
$ module list
Currently Loaded Modulefiles:
1) ccc           3) dfldatadir/own   5) extenv/shspace
2) datadir/own   4) datadir/shspace  6) thetool/2
```

Next section will give some hints to write your own modulefiles.

9.5.2 Write your own modulefile

This section describes how to write a modulefile through the example of a product installed in the shared space named *shspace*. This product is called *TheCode* and it is installed in version *1.2.3*. This product depends on the library *libprod* and on configuration enabled via the *conf/thecode* modulefile. It is advised to install this product in the `$SHSPACE_PRODUCTSHOME/thecode-1.2.3` directory.

Now looking at modulefiles, we suggest to have one modulefile for each version of the product and one common modulefile referred by all version-specific modulefiles which contains all the definition relative to the product. In our example for *TheCode* product, it means having first a version-specific modulefile at `$SHSPACE_MODULEFILES/thecode/1.2.3` containing:

```
#%Module1.0

# Software description
set version "1.2.3"

# load common functions and behavior
source $env(SHSPACE_MODULEFILES)/thecode/.common
```

In this version-specific modulefile, we just set the version number then we load the common definitions for the product. By doing so, product definition is easily shared between the different versions available of this product. Moving on the common modulefile at `$SHSPACE_MODULEFILES/thecode/.common`:

```
#%Module1.0

# Software description
set whatis       "TheCode"
set software     "thecode"
set description "One sentence to describe what TheCode is done for"

# Conflict
conflict $software
# Prerequisite
prereq   conf/thecode
prereq   libprod

# load head common functions and behavior
source $env(MODULEFILES)/.scripts/.headcommon

# Loads software's environment
# application-specific variables
set prefix       "$env(SHSPACE_PRODUCTSHOME)/$software-$version"
set libdir       "$prefix/lib"
set incdir       "$prefix/include"
# compilerwrappers-specific variables
set ldflags      "<ldflags>"

append-path PATH            "$bindir"
append-path LD_LIBRARY_PATH "$libdir"

setenv VARNAME "VALUE"

# load common functions and behavior
```

```
source $env(MODULEFILES)/.scripts/.common
```

This common modulefile first defines the identity card of the product, by setting the local variables `software`, `description`, etc. Conflicts and prerequisites are then setup with the `conflict` and `prereq` keywords. Some computing center global definitions are then loaded, and also at the end of the file. These definitions help to get the same behavior for your modulefiles as for the computing center regular modulefiles, like for instance the message printed at **module load** and also the structured description returned when calling **module help** on a product.

Then special local variables are set to define the environment of the software. Variables `prefix`, `libdir`, `incdir`, `cflags`, `ldflags` will automatically create the environment variables `THECODE_ROOT`, `THECODE_LIBDIR`, `THECODE_INCDIR`, `THECODE_CFLAGS` and `THECODE_LDFLAGS` when the modulefile is loaded. These environment variables are guessed and set by the global common file `$MODULEFILES/.common` sourced at the end of this application-specific common file `$SHSPACE_MODULEFILES/thecode/.common`. Environment variables needed by the software can also be defined or ajusted as it is done in the example for the `VARNAME`, `PATH` and `LD_LIBRARY_PATH` variables.

Note: Modulefile inter-dependencies are managed automatically by **module**. Any time the modulefile is loaded its dependencies, defined with the `prereq` keyword, are automatically loaded.

We recommend the use of the previous example as a template for all your modulefiles. If you use the recommended path for all product installations, you can keep major parts of this template as it is. Just specify the right software name and version, the correct dependencies and the product-specific environment variables like `VARNAME` in the example. Using that template will ensure you that your modulefile behaves the same way as the default modules and that all the available **module** commands will work for you.

To further learn the syntax of a modulefile, please refer to the `modulefile` and `module` man pages. Moreover a modulefile is interpreted by the **module** command as a TCL script, so you can use the TCL code and functions. For more information on the TCL language, please refer to http://tcl-lang.org/doc/.

9.5.3 Improve your Modules performances with cache (Advanced Usage)

The `cachebuild` sub-command of the module system creates a cache file in module paths (`.modulecache`). Without arguments, it attempts to create cache in every enabled modulepath where the running user has write access. If arguments are provided, the cache is built in the directories specified by these arguments.

When dealing with environments that utilize a large number of module files, filesystem performance issues can arise. Invoking a module command scans the modulepath for available modules, a process that can significantly delay operations in systems with extensive module collections. This scanning puts a strain on the filesystem with a high volume of read operations, particularly in shared or high-performance computing environments. The cumulative effect of multiple users executing such operations can further exacerbate the issue.

The `module cachebuild` command mitigates these challenges by generating a cache file for the module paths. This cache acts as a snapshot of available modules, enabling the module command to quickly identify the available modules without scanning the entire modulepath. Reducing the need for extensive directory reads significantly decreases the time required to initialize module environments and execute module-related operations, thus improving overall system performance and user experience.

Additionally, by lessening the impact on the filesystem, `module cachebuild` aids in maintaining system responsiveness and stability, especially important in multi-user environments where filesystem performance is crucial. Users are advised to regularly update the module cache, particularly after adding or updating modules, to ensure that the cache accurately represents the current state of the module paths.

1. To run without any arguments, simply execute:

```
module cachebuild
```

This command attempts to create a cache file in every enabled module path where the running user has write permissions.

2. To build the cache in specific directories, provide the paths as arguments:

```
module cachebuild /path/to/custom/modulepath1 /path/to/custom/modulepath2
```

This builds the cache only in the specified directories, offering more control over the cache generation process.

3. A good practice with `extenv`:

```
module cachebuild $SHSPACE_MODULEFILES
```

This builds the cache of `$SHSPACE_MODULEFILES`.

Regularly using `module cachebuild` can significantly enhance the responsiveness of module operations, particularly in environments with a large assortment of modules. It presents a straightforward yet effective strategy to mitigate filesystem-related performance issues, making it a strongly recommended practice for users.

Warning: After any modification to modules (such as adding, removing, or updating a module), it is crucial to rerun `module cachebuild` to update the cache accordingly. Failing to do so may result in discrepancies between the cache and the actual available modules.

SOFTWARES

This section presents the various kinds of software products (binaries, libraries, scripts and tools) available on the center. It is organized as follows. After generally explaining the different kinds of software on the center, we will focus on specific cases (restricted and purchased products). Finally, we will talk about the software life cycle, from the installation request to the product removal.

10.1 Generalities on software

Lots of tools and products are available on the supercomputer. A distinction is made between the OS and CCC software.

10.1.1 OS software

They are software products provided by the operating system. They are installed by packages (rpm) and are locally stored on each node.

All common Linux software products enter in this category, such as:

- shells (**bash**, **tcsh**),
- common scripting tools (**cat**, **sed**, **awk**, ...)
- editors (**vi**, **emacs**, **gedit**, ...),
- minimal installations of common scripting language (**perl**, **python**).

Updates occur during maintenance or during production only if they do not impact the operating system of the node.

10.1.2 CCC software

CCC software products are the third-party dedicated tools or dedicated compilations. They are shared by all center nodes and are installed by TGCC-CCRT dedicated service(s) during production. We prefer the term CCC software products over center software which is ambiguous.

Contrary to the OS software products, CCC software products are not within the standard operating system paths for binaries or libraries (eg. `/usr/bin` or `/usr/lib64`).

Each product is installed in a dedicated directory and requires an update of environment variables like and before use. These environment updates are handled by the **module** tool detailed in the *Environment management* section.

Updates occur during production as they do not interrupt nor interfere with the operating system installation.

10.1.3 Software and module

A CCC software product requires an environment update before use. The **module** command is the tool for it, so a module exists for each CCC software product.

An OS software product is usually not referred to by a module, except when:

- it is usually the case in other computing centers (ex: **tcl**)

- there is a CCC software version of it (ex: **valgrind**): it helps to avoid conflicts between the OS and CCC installation.

Products referenced by **module** have been organized into categories:

- applications: simulation products

- environment: defines the user/group environment

- tools: development tools

- graphics: visualization or image manipulation tools

- parallel: parallel execution software such as MPI

- libraries: third-party libraries

- compilers: the various compilers

A domain-oriented view is provided by **module search <keyword>**. To list the valid keywords use:

```
module help|show products/keywords
```

10.1.4 Software toolchains

> **Warning:** Please refer to internal technical documentation to get information about available toolchains.

This approach provides a variety of compilation / runtime environments to users with consistency across products. To consult the list of available builds for a given product, type **module help <product>**.

The selection between toolchains is made with two modules flavor/buildcompiler/<compiler>/<version> and flavor/buildmpi/<mpi>/<version>. It allows you to change the toolchain used for any loaded product regardless of the Intel compiler and MPI implementation used at runtime, which is determined by your current intel and mpi modules loaded.

More information on the flavor mechanism can be found on the *dedicated section*.

The module flavor/buildcompiler/<compiler>/<version> lets you choose the version of the Intel compiler whereas the module flavor/buildmpi/<mpi>/<version> lets you choose the MPI implementation and version.

10.2 Product Life cycle

Here are the phases of a software product life cycle, explained chronologically from its installation request to its deprecation and removal.

10.2.1 Installation request

New products and/or product update(s) can be requested by mail to the TGCC-CCRT hotline.

Requests are usually accepted if:

- the product is the last stable version,
- the product is useful to many users,
- the product has no license issue.

There are also special cases, namely:

- an installation for a group of persons is not possible since these persons are not part of the same user container,
- or the product access should be restricted (see *Installation request and restricted products*),
- or the product requires a license server (see *Installation request and purchased products*).

10.2.2 Installation phase

If the installation request is approved, we will proceed to the installation itself:

- for an OS software product, it should be done at the next maintenance (or before if this is deemed safe).
- and for an CCC software product, it should take several days.

Note: Those are estimated installation times. If technical difficulties occur the installation may take longer (example: licensed products)

Newly installed CCC software maybe listed with:

```
module help|show products/newinstall
```

10.2.3 Production phase

During the production, the software product should be operational. If you detect an issue, a bug or a missing feature, please report it to the TGCC-CCRT hotline.

10.2.4 End-of-life and removal

An OS software product is considered obsolete when either the distribution considers it as such or when an update has been planned for the next maintenance.

For CCC software products, we try to respect the following rules. For each CCC software product, the center favors versions with the least bugs, the most features and the best performances. Hence the last stable version of a product is very welcome (new features, better performances and bug corrections). So when a new version of a product is installed, previous versions may become obsolete. To be more specific, we will only keep:

- the latest version,

- its last known stable(s) version(s) (for reference purpose),

- and the version(s) before retro-compatibility issues (mainly API break).

10.3 Python

10.3.1 Available versions and toolchains

To display all Python versions, use the command `module av -t python3`:

```
python3/x.y.z
python3/x.y.z(stable)
python3/x.y.z(unstable)
```

The *(stable)* version only changes for security issues. The *(unstable)* version changes everytime a user asks for a new package or an update. We higly recommand the *(stable)* one for **virtualenv** users.

There are at least three types of Python distributions:

- **System:**

    - Only packages that could be compiled with the operating system GCC

- **GCC+OpenMPI:**

    - Compiled with a recent GCC and Open MPI

- **GCC+OpenMPI+CUDA**

    - Compiled with a recent GCC, Open MPI and CUDA

    - For AI applications (PyTorch, Tensorflow. . .)

Display available configurations with the command `module help python3/<version>`.

```
------------------------------------------------------------------
 Module Specific Help for /ccc/etc/modulefiles/tools/python3/<version>:

   Software description:
     Name         : Python lang
     Description   : Python programming language
     Version       : <versions>

   Software configuration(s):

     Copy/paste one of the following line ("module purge" could be required).
```

(continues on next page)

```
    Setup(s) for users based on GCC compiler:

      0 : module load gcc/<versions> mpi/openmpi/<versions> flavor/python3/cuda-
→<version> python3/<version>
      1 : module load gcc/<versions> mpi/openmpi/<versions> python3/<version>

    Setup(s) without specific requirement:

      2*: module load python3/<version>


  ------------------------------------------------------------------
```

For instance, to load the GCC+Openmpi+CUDA distribution:

```
$ module load gcc/<version> mpi/openmpi/<version> flavor/python3/cuda-<version> python3/
→<version>
```

To list all available Python packages in your current environment, run *pip3 list*.

10.3.2 Adding new module

Adding new modules in the distribution

In order to request the installation of one or several python modules, please refer to the Installation request part.

Adding new modules locally, through a virtual environment

Load the desired Python environment:

```
$ module purge ; module load [...] python3/<version>
```

Create a virtual environment based on this environment:

```
$ python3 -m venv <my_virtual_env> --system-site-packages
```

--system-site-packages option indicates that the original Python site-packages will be available to the new environment. Without this option, the new virtual environment will be empty and the environment should be rebuilt from scratch.

Activate it:

```
source <my_virtual_env>/bin/activate
```

Import the source code of the Python modules you need to install, for instance: maze-3.0.0.tar.gz (see *Data transfers*).

Extract the folder from the archive:

```
tar xfz <module.tar.gz>
```

Go in the folder containing the installation files:

```
cd <module>
```

Install the python module:

```
python3 setup.py install
```

You may need to install one or several dependencies to complete the installation. Proceed in the same way for each of those. It is recommended to identify beforehand the aforementioned dependencies either on a local computer and/or by reading the requirements.txt file that should be at the root of the Python package.

10.3.3 Machine learning and artificial intelligence

Pytorch

In order to use Pytorch, you may use the following environment:

```
$ module purge
$ module load gcc/<version> mpi/openmpi/<version> flavor/python3/cuda-<version> python3/
↪<version>
$ python3
Python 3.x.y (main, XXX XX 20XX, XX:XX:XX) [GCC X.Y.Z] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>>
```

Tensorflow

In order to use Tensorflow, you may use the following environment:

```
$ module purge
$ module load gcc/<version> mpi/openmpi/<version> flavor/python3/cuda-<version> python3/
↪<version>
$ python3
Python 3.x.y (main, XXX XX 20XX, XX:XX:XX) [GCC X.Y.Z] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow
>>>
```

10.3.4 Jupyter-lab

Jupyter-lab is available on irene. You may use the following commands to start Jupyter-lab:

```
$ module purge
$ module load python3/<version>
# or module load gcc/<version> mpi/openmpi/<version> python3/<version>
# or module load gcc/<version> mpi/openmpi/<version> flavor/python3/cuda-<version>␣
↪python3/<version>
$ jupyter-lab
```

> **Warning:** Jupyter-lab is not available with the Intel toolchain.

It is recommended to use NiceDCV for better stability.

10.4 Alphafold

We provide access to the DeepMind's Alphafold software via a container. Included are the OCI image, the complete database and a set of scripts for easier usage.

10.4.1 Available versions

The version available is 2.3.1, and the database includes the latest versions of all files as of December 2022.

10.4.2 Using AlphaFold on TGCC

AlphaFold requires access to the **|default_GPU_partition|** partition. It may works on other GPU partition. Upon loading the module, you will receive a path to our AlphaFold Readme file.

```
$ module load alphafold
```

First import the image using pcocc-rs.

```
$ pcocc-rs image import docker-archive:${ALPHAFOLD_IMAGE_ROOT}/alphafold-2022.10.24.tar
→user:alphafold-231
```

Next, please copy the scripts we have provided to utilize AlphaFold via pcocc-rs.

```
$ cp -r ${ALPHAFOLDRUN_ROOT}/run_squashfs-flat ${CCCSCRATCHDIR}/<my_test_case>
```

You have the option to select between monomer or multimer cases. This can be configured in the submission script, with monomer being the default setting. The output directory should exist before sumbitting the script.

```
$ cat ${ALPHAFOLDRUN_ROOT}/run_squashfs-flat/submit_script_alphafold-231_squashfs-flat.sh
#!/bin/bash
#MSUB -e AF_%J.e
#MSUB -o AF_%J.o

#MSUB -n 1 # nombres de tâches MPI
#MSUB -c |cores_per_GPU_on_default_partition| # number of cores to be booked. You need
→|cores_per_GPU_on_default_partition| cores to get one GPU.
##MSUB -x #node exclusivity, to be add for big cases. (if you encounter a memory issue)

#MSUB -T 20000
#MSUB -q |default_GPU_partition|    # hybrid may be used as well
#MSUB -A <group>
#MSUB -m work,scratch

module purge
module load alphafold
export CCC_ALPHAFOLD_DATABASE
export CCC_ALPHAFOLD_IMAGE="alphafold-231"

# Use the database stored on hdd to test the impact
#export ALPHAFOLD_USE_DB_HDD=true

# the output directory should exist before launching the job. Adapt to your need.
```

(continues on next page)

```
export ALPHAFOLD_OUTPUT="$(pwd)/output-squash-flat"

# to edit to match your need
export ALPHAFOLD_INPUT=${ALPHAFOLDRUN_ROOT}/exemple_CCRT/input.fasta

# Replace ${ALPHAFOLDRUN_ROOT}/run_pcocc-rs_squashfs-flat_|default_GPU_partition|.sh by␣
↪the path to your submission script if you want to use a tailored version
ccc_mprun ${ALPHAFOLDRUN_ROOT}/run_squashfs-flat/run_pcocc-rs_squashfs-flat_|default_GPU_
↪partition|_monomer.sh --max_template_date 2020-05-14 --fasta_paths $ALPHAFOLD_INPUT >␣
↪AF-231-squash-flat.log
```

Finally, simply submit the script as usual.

```
$ ccc_msub submit_script_alphafold-231_squashfs-flat.sh
```

The database includes the following files.

```
/ccc/products/alphafold-db-2022.10.24/system/default/db/
├── alphafold-db-compressed_hdd.sqsh
├── alphafold-db-compressed.sqsh
├── alphafold-db-flat_hdd.sqsh
├── alphafold-db-flat_t1.sqsh
├── bfd
│   ├── bfd_metaclust_clu_complete_id30_c90_final_seq.sorted_opt_a3m.ffdata
│   ├── bfd_metaclust_clu_complete_id30_c90_final_seq.sorted_opt_a3m.ffindex
│   ├── bfd_metaclust_clu_complete_id30_c90_final_seq.sorted_opt_cs219.ffdata
│   ├── bfd_metaclust_clu_complete_id30_c90_final_seq.sorted_opt_cs219.ffindex
│   ├── bfd_metaclust_clu_complete_id30_c90_final_seq.sorted_opt_hhm.ffdata
│   └── bfd_metaclust_clu_complete_id30_c90_final_seq.sorted_opt_hhm.ffindex
├── mgnify
│   └── mgy_clusters.fa
├── params
│   ├── LICENSE
│   ├── params_model_1_multimer_v3.npz
│   ├── params_model_1.npz
│   ├── params_model_1_ptm.npz
│   ├── params_model_2_multimer_v3.npz
│   ├── params_model_2.npz
│   ├── params_model_2_ptm.npz
│   ├── params_model_3_multimer_v3.npz
│   ├── params_model_3.npz
│   ├── params_model_3_ptm.npz
│   ├── params_model_4_multimer_v3.npz
│   ├── params_model_4.npz
│   ├── params_model_4_ptm.npz
│   ├── params_model_5_multimer_v3.npz
│   ├── params_model_5.npz
│   └── params_model_5_ptm.npz
├── pdb70
│   ├── md5sum
│   ├── pdb70_a3m.ffdata
│   ├── pdb70_a3m.ffindex
```

```
│   ├── pdb70_clu.tsv
│   ├── pdb70_cs219.ffdata
│   ├── pdb70_cs219.ffindex
│   ├── pdb70_hhm.ffdata
│   ├── pdb70_hhm.ffindex
│   └── pdb_filter.dat
├── pdb_mmcif
│   ├── mmcif_files
│   └── obsolete.dat
├── pdb_seqres
│   └── pdb_seqres.txt
├── uniclust30
│   ├── UniRef30_2022_02_a3m.ffdata
│   ├── UniRef30_2022_02_a3m.ffindex
│   ├── UniRef30_2022_02_cs219.ffdata
│   ├── UniRef30_2022_02_cs219.ffindex
│   ├── UniRef30_2022_02_hhm.ffdata
│   ├── UniRef30_2022_02_hhm.ffindex
│   └── UniRef30_2022_02.md5sums
├── uniprot
│   ├── uniprot_sprot.fasta
│   └── uniprot_trembl.fasta
└── uniref90
    └── uniref90.fasta
```

Note: For advanced use of AlphaFold, you may modify run_pcocc-rs_squashfs-flat_|default_GPU_partition|_monomer.sh and run_pcocc-rs_squashfs-flat_|default_GPU_partition|_multimer.sh as needed.

Note: AlphaFold is accessible to members of the alphafold UNIX group. To join, please contact support at please refer to internal documentation to get hotline email or please refer to internal documentation to get hotline phone.

10.5 DMTCP

The DMTCP (Distributed MultiThreaded Checkpointing) tool is used to transparently checkpoint the state of HPC applications, including multi-threaded applications.

Warning: Let be mentioned that MPI is not supported in the currently installed DMTCP versions.

10.5.1 Usage

In order to run an application with DMTCP, a server is launched with dmtcp_coordinator in advance. Then, the DMTCP run command dmtcp_launch is used to call the user application. DMTCP creates checkpoints, which can be relaunched after exiting the run. This procedure is explained step-by-step in the following tutorial.

10.5.2 Tutorial

For this tutorial, a simple application that counts from 1 to 100 is introduced with the following C code file:

```
$ cat test_dmtcp.c
#include <stdio.h>
#include <unistd.h>
#define MAX 100

int main () {
        for (int i = 1; i <= MAX; i++) {
                sleep(1);
                printf("%d/%d\n", i, MAX);
        }
}
```

Interactive launch

In order to test the tutorial example, please open an interactive session. Use for example the following interactive allocation:

```
$ ccc_mprun -p <partition> -m work -c 1 -K
```

Load the DMTCP module and compile the example application:

```
$ gcc test_dmtcp.c
$ module load dmtcp
```

Run the server through the `dmtcp_coordinator` command. It is recommended to use the daemon mode (`--daemon`) and option `--exit-on-last`, which helps to terminate the daemon properly. Furthermore, option `-i` is used to indicate the frequency (in seconds) of the checkpoints.

```
$ dmtcp_coordinator --daemon --exit-on-last -i 2
dmtcp_coordinator starting...
    Host: xxxx.xxxx.xxx.xxx.xxx.xxx.xx (xx.xx.x.x)
    Port: 7779
    Checkpoint Interval: 2
    Exit on last client: 1
[2024-10-25T15:53:56.035, 4150687, 4150687, ]Backgrounding...
```

Run the application and stop it at an arbitrary moment with `ctrl+C`:

```
$ dmtcp_launch -j ./a.out
1/100
2/100
3/100
^C
```

In the present example, a checkpoint file is created every two seconds (`-i 2` above). The application can be restarted through an automatically generated restart script.

Restart the application:

```
$ ./dmtcp_restart_script.sh
3/100
4/100
5/100
^C
```

Launch with job files

This second example shows how to launch the previous code example with a job file. The job file writes:

```
$ cat job.sh
#!/bin/bash

#MSUB -c 1
#MSUB -n 1
#MSUB -Q test
#MSUB -T 60
#MSUB -q <partition>
#MSUB -m work

module purge

module load dmtcp

dmtcp_coordinator --daemon --exit-on-last -i 2

ccc_mprun -E "-u" dmtcp_launch -j ./a.out
```

As usual, the job can be run with the `ccc_msub` command, and the program a.out will be stopped at the end of the job. If needed, the `ccc_mdel` command can be used to kill a job, which also stops the program. Note that `-E "-u"` sets the output as unbuffered, in order to save it to the output files `*.o`.

To run the restart on the previously used compute node, the environment variable `HOSTNAME` is used and the restart job script writes:

```
$ cat job_restart.sh
#!/bin/bash

#MSUB -c 1
#MSUB -n 1
#MSUB -Q test
#MSUB -T 60
#MSUB -q <partititon>
#MSUB -m work

export DMTCP_COORD_HOST=$HOSTNAME

ccc_mprun -E "-u" ./dmtcp_restart_script.sh
```

10.6 Products list

This section catalogs all the software accessible on irene. For further details about these offerings, please engage with our support team. A number of applications within this list have the capability to execute computations on GPUs, as denoted by the "GPU Support" column. To clarify, this designation is exclusive to end-user applications. However, this does not preclude the possibility of other categories of products, such as tools and libraries, from being capable of running on GPUs. Should you require a GPU-aware version of any application not currently indicated as such, we encourage you to reach out to the support team for support.

The list is available in *the dedicated section*.

10.7 Specific software

The center has specific rules for restricted products and purchased products.

10.7.1 Restricted software products

A restricted product is referenced by **module**, but its access is restricted to a specific Unix group. Although the reasons may vary, this usually comes from a license requirement.

For example, a licensed product may restrict its usage to academic purposes, or the purchased license is an academic one.

> **Warning:** Please refer to internal technical documentation to get information about this subject.

Installation request and restricted products

> **Warning:** Please refer to internal technical documentation to get information about this subject.

10.7.2 Products with purchased license

Some products (such as Intel compilers) require a license server. We refer to them as *licensed products*.

Module and licensed products

Here are several adaptations of **module** for *licensed products*:

- A *licensed product module* depends on a *license module* to find its license server.
- Although the *license module* is not shown at load time, a **module list** can display it.
- *license module* are named *licsrv/<server>* and are listed by:

```
module avail licsrv
```

Installation request and purchased products

While requesting the installation of a *purchased product*, you may:

- have a license not requiring a dedicated server,

- have the license and want to restrict its usage to a group of people (because you only have a few tokens),

- have a global license (limited in time),

- have no license and ask for the center to purchase it.

The two first cases are handled by defining it as a *restricted product*.

The last case must be clearly justified. If so, you must at least:

- estimate the number of user,

- specify its application domain,

- tell us if you need the academic or the commercial license,

- and tell us if you need a global license and/or how many tokens are necessary.

Please note that, even if such installation follows the usual validation process, you may have additional delays, namely:

- if you need a non-FlexNet dedicated license server,

- if you need to generate some license server identification (done by us and filtered for security reasons) before getting the license.

PRODUCTS LIST

11.1 Applications

Table 1: Applications products list

Name	Description	GPU Support
abinit	ABINIT is a package whose main program allows one to find the total energy, charge density and electronic structure of systems made of electrons and nuclei	
agate	Agate is mainly designed to work with the Abinit DFT code. However, it can perfectly read some VASP files and other commonly used format. Agate should help you to visualize in a glance any input file to make sure the structure you're simulating is the one you want.	
alphafold-db	Alphafold database	
alphafold-img	Alphafold container image	
alphafold-run	Alphafold wrapper to ease its use	
alphafold	AI predicting protein folding structure	✓
aster	Code_Aster is a thermomechanics and analysis of structure package	
cantera	Cantera is an open-source suite of tools for problems involving chemical kinetics, thermodynamics, and transport processes.	
codee	Codee is a development tool that helps you enhance your code by following the recommendations of the first Open Catalog of Best Practices for Modernization and Optimization	
conquest	CONQUEST is a large scale DFT electronic structure code, capable of both diagonalisation and linear scaling, or O(N), calculations. It is developed jointly by NIMS (National Institute for Materials Science, Japan), ISM (Institut des Science Moleculaires, University of Bordeaux) and UCL.	
cp2k	CP2K performs atomistic and molecular simulations of solid state, liquid, molecular and biological systems	
cst-studio	high-performance 3D EM analysis software package for designing, analyzing and optimizing electromagnetic (EM) components and systems.	
espresso	Quantum ESPRESSO is an integrated suite of computer codes for electronic-structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves, and pseudopotentials (both norm-conserving and ultrasoft).	
fds	Fire Dynamics Simulator	

Table 1 – continued from previous page

Name	Description	GPU Support
freefem	FreeFem is an implementation of a language dedicated to the finite element method. It enables you to solve Partial Differential Equations (PDE) easily.	
galilee-v0-lux		
galilee-v0		
gate	GATE is a Monte-Carlo simulation toolkit for medical physics applications	
geant4	Toolkit for the simulation of the passage of particles through matter	
gmt	Generic Mapping Tools	
gromacs	GROMACS (GROningen MAchine for Chemical Simulations) is a molecular dynamics simulation package	✓
lammps	LAMMPS is a molecular dynamics code, an acronym for Large-scale Atomic/Molecular Massively Parallel Simulator	✓
ls-prepost	LS-PrePost is an advanced pre and post-processor that is delivered free with LS-DYNA. The user interface is designed to be both efficient and intuitive. LS-PrePost runs on Windows, Linux, and Macs utilizing OpenGL graphics to achieve fast rendering and XY plotting.	
mpcci	The MpCCI interface software is a vendor neutral solution for co-simulation and file-based data transfer. MpCCI supports a growing number of commercial as well as research simulation tools in different engineering disciplines.	
mstar	Build advanced fluid models in minutes, predict real-time dynamics with precision, and solve complex fluid flow problems faster. M-Star is CFD software for the real world.	
openfoam-plus	OpenFOAM distributed by OpenCFD (Open Source suite for field operations and manipulations).	
openfoam	Open Source suite for field operations and manipulations.	
particleworks	Simulates liquid behaviors such as water and oil, at high-speed.	
permas	PERMAS is a general purpose software system to perform complex calculations in engineering using the finite element method (FEM), and to optimize the analyzed structures and models.	
prolb	ProLB is an innovative Computational Fluid Dynamics (CFD) software solution. Based on the Lattice-Boltzmann method, its succesfully-validated solver performs inherently transient simulations of highly complex flows with a competitive turnaround time. ProLB??s accurate aerodynamic and aeroacoustic modeling allows engineers to make early design decisions that optimize and shorten the product development process.	
qagate	qAgate is the new Qt interface for agate. It includes all the features of Agate with a new user friendly look. It also includes sommes 'external' tools to help some users with fast analysis.	
salome	SALOME is an open-source software that provides a generic platform for Pre- and Post-Processing for numerical simulation. It is based on an open and flexible architecture made of reusable components.	
saturne	Code_Saturne is a EDF's general purpose Computational Fluid Dynamics (CFD) software	
siesta	Spanish Initiative for Electronic Simulations with Thousands of Atoms	

Table 1 – continued from previous page

Name	Description	GPU Support
taitherm	AITherm® is a 3D thermal simulation software that predicts temperatures using transient or steady-state analysis. TAITherm thermal analysis software is as easy to use as it is powerful. It solves for solar and thermal radiation, convection, and conduction, including ever-changing environments. TAITherm removes the burden of complicated thermal simulations and ensures that your design will function optimally in real-world scenarios.	
tetramax	Synopsys TestMAX™ ATPG is Synopsys' state-of-the-art pattern generation solution that enables design teams to meet their test quality and cost goals with unprecedented speed.	
wps	The Weather Research and Forecasting (WRF) Model is a next-generation mesoscale numerical weather prediction system designed to serve both atmospheric research and operational forecasting needs. WPS is the WRF Preprocessing System.	
wrf	The Weather Research and Forecasting (WRF) Model is a next-generation mesoscale numerical weather prediction system designed to serve both atmospheric research and operational forecasting needs	
yambo	Yambo is a FORTRAN/C code for Many-Body calculations in solid state and molecular physics.	

11.2 Compilers

Table 2: Compilers products list

Name	Description
aocc-compiler	The AOCC compiler system
aocl-libm	The AOCL AMD Math Library
c++/gnu	GNU C++ compiler (g++)
c++/intel	Intel C++ compiler (icpc)
c++/inteloneapi	Intel oneAPI C++ compiler (icpx)
c++/llvm	LLVM C++ compiler (clang++)
c++/nvidia	Nvidia C++ compiler (nvc++)
c++/pgi	PGI C++ compiler (pgCC)
c/gnu	GNU C compiler (gcc)
c/intel	Intel C compiler (icc)
c/inteloneapi	Intel oneAPI C compiler (icx)
c/llvm	LLVM C compiler (clang)
c/nvidia	Nvidia C compiler (nvc)
c/pgi	PGI C compiler (pgcc)
fortran/gnu	GNU Fortran compiler (gfortran)
fortran/intel	Intel Fortran compiler (ifort)
fortran/inteloneapi	Intel oneAPI Fortran compiler (ifx)
fortran/llvm	LLVM Fortran compiler (flang)
fortran/nvidia	Nvidia FORTRAN compiler (nvfortran)
fortran/pgi	PGI Fortran compiler (pgf77/pgf90)
gnu	GNU C (gcc), C++ (g++) and Fortran (gfortran) compilers
intel	Intel C (icc), C++ (icpc) and Fortran (ifort) compilers
inteloneapi	Intel oneAPI C (icx), C++ (icpx), DPC++ (dpcpp) and Fortran (ifx) compilers
java/openjdk	Open Java Runtime and Development Kit
java/oracle	Oracle Java Runtime and Development Kit
llvm	The LLVM Project is a collection of modular and reusable compiler (Clang, Clang++) and toolchain technologies.
nvhpc	Nvidia C (nvc), C++ (nvc++) and Fortran (nvfortran) compilers

11.3 Graphics

Table 3: Graphics products list

Name	Description
ensight	Post-processing and Visualization for Scientific Data
ferret	Software for vizualisation and data analysing
ghostscript	An interpreter for the PostScript language and for PDF, and related software and documentation
gnuplot	Gnuplot is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms.
grace	Visualization software
hdfview	HDFView is a visual tool for browsing and editing HDF4 and HDF5 files.
idl	IDL is a solution for data analysis, data visualization, and software application development
imagemagick	Toolkit for manipulation of graphic images, including conversion of images between a variety of different formats
libpng	libpng is the official PNG reference library.
mesa	Mesa libGL/libGLU libraries
ncl_ncarg	NCAR Graphics is primarily a graphics package, with some limited data analysis through the Ngmath package.
ncview	Ncview is a visual browser for netCDF format files
paraview	Open source scientific visualization
ploticus	Ploticus is a free GPL software utility that can produce various types of plots and graphs.
qt	Cross-platform application and UI development framework
tecplot	Visualization software tool
visit	VisIt - Visualization Tool
vmd	VMD is a molecular visualization program for displaying, animating, and analyzing large biomolecular systems using 3-D graphics and built-in scripting.
vtk	The Visualization Toolkit (VTK) is an open-source, freely available software system for 3D computer graphics, image processing and visualization

11.4 Libraries

Table 4: Libraries products list

Name	Description
adge	
amd-fftw	Fastest Fourier Transform in the West
aocl-libmem	AOCL-LibMem is AMD's optimized implementation of memory/string functions.

Table 4 – continued from previous page

Name	Description
aocl-rng	AMD Random Number Generator Library is a pseudorandom number generator library. A pseudo-random number generator (PRNG) produces a stream of variates that are independent and statistically indistinguishable from a random sequence. AMD Random Number Generator Library provides a comprehensive set of statistical distribution functions which are founded on various underlying uniform distribution generators (base generators) including Wichmann-Hill and Mersenne Twister. The library contains five base generators and twenty-three distribution generators. In addition users can supply a custom built generator as the base generator for all of the distribution generators.
aocl-securerng	The AMD Secure Random Number Generator (RNG) is a library that provides APIs to access the cryptographically secure random numbers generated by AMD's hardware-based random number generator implementation. These are high quality robust random numbers designed to be suitable for cryptographic applications. The library makes use of RDRAND and RDSEED x86 instructions exposed by the AMD hardware. Applications can just link to the library and invoke either a single or a stream of random numbers. The random numbers can be of 16-bit, 32-bit, 64-bit or arbitrary size bytes.
aocl-sparse	AOCL-Sparse contains basic linear algebra subroutines for sparse matrices and vectors optimized for AMD processors. It is designed to be used with C and C++.
aocl	AMD Optimizing CPU Libraries (AOCL)
apr-util	Apache Portable Runtime libraries
apr	Apache Portable Runtime libraries
armadillo	Armadillo is a high quality C++ linear algebra library, aiming towards a good balance between speed and ease of use
blas/aocl	Basic Linear Algebra Subprograms
blas/blis	BLIS is a portable software framework for instantiating high-performance BLAS-like dense linear algebra libraries
blas/mkl	Intel Math Kernel Library BLAS routines
blas/netlib	Netlib Basic Linear Algebra Subprograms
blas/openblas	OpenBLAS is an optimized BLAS library based on GotoBLAS2 1.13 BSD version.
blis	BLIS is a portable software framework for instantiating high-performance BLAS-like dense linear algebra libraries
blis/aocl	BLIS is a portable software framework for instantiating high-performance BLAS-like dense linear algebra libraries
blitz	Blitz++ provides dense arrays and vectors, random number generators, and small vectors (useful for representing multicomponent or vector fields).
boost	Boost C++ libraries
cgal	CGAL is a software project that provides easy access to efficient and reliable geometric algorithms in the form of a C++ library.
cgns	CFD General Notation System
cudnn	The NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks.
cwipi	A dynamic parallel code coupler and coupling with parallel interpolation interface
czmq	High-level C binding for ZeroMQ
dbcsr	DBCSR is a library designed to efficiently perform sparse matrix-matrix multiplication, among other operations. It is MPI and OpenMP parallel and can exploit Nvidia and AMD GPUs via CUDA and HIP.

Table 4 – continued from previous page

Name	Description
dtcmp	The Datatype Comparison (DTCMP) Library provides pre-defined and user-defined comparison operations to compare the values of two items which can be arbitrary MPI datatypes.
eccodes	ecCodes provides an application programming interface and a set of tools for decoding and encoding messages in GRIB format
editline	A small replacement for GNU readline() for UNIX
eigen	Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms
elpa	Eigenvalue soLvers for Petaflop Applications
fftw2	Discrete Fourier Transform
fftw3	Fastest Fourrier Transform in the West
fftw3/aocl	Fastest Fourier Transform in the West
fftw3/mkl	Intel Math Kernel Library FFTW3 routines
fftw3/vanilla	Fastest Fourier Transform in the West
fltk	FLTK is a cross-platform C++ GUI toolkit for UNIX/Linux
fmt	fmt (formerly cppformat) is an open-source formatting library. It can be used as a fast and safe alternative to printf and IOStreams.
fox	A Fortran library for XML
gdal	gdal is a translator library for raster geospatial data formats that is released under an X/MIT style
geos	Geometry Engine Open Source
glpk	The GLPK (GNU Linear Programming Kit) package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems
gmp	GMP is a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating-point numbers.
grib	The ECMWF GRIB API is an application program interface accessible from C and FORTRAN programs developed for encoding and decoding WMO FM-92 GRIB edition 1 and edition 2 messages.
gsl	GNU Scientific Library (GSL) is a numerical library for C and C++ programmers
h5utils	h5utils is a set of utilities for visualization and conversion of scientific data in the free, portable HDF5 format.
hdf5	Hierarchical Data Format
hypre	Hypre is a library for solving large, sparse linear systems of equations on massively parallel computers
ipp	Intel Integrated Performance Primitives Library
jansson	C library for encoding, decoding and manipulating JSON data
jasper	Common jpeg library
kokkos-kernels	KokkosKernels implements local computational kernels for linear algebra and graph operations, using the Kokkos shared-memory parallel programming model.
kokkos-tools	Performance profiling tools that can be optionally integrated into Kokkos-based applications and libraries.
kokkos	Kokkos Core implements a programming model in C++ for writing performance portable applications targeting all major HPC platforms. Kokkos Core is part of the Kokkos C++ Performance Portability Programming EcoSystem, which also provides math kernels (kokkos-kernels), as well as profiling and debugging tools (kokkos-tools).
lapack/aocl	Linear Algebra PACKage
lapack/mkl	Intel Math Kernel Library LAPACK routines
lapack/netlib	Netlib Linear Algebra PACKage

Table 4 – continued from previous page

Name	Description
lapack/openblas	OpenBLAS is an optimized BLAS library based on GotoBLAS2 1.13 BSD version.
libaec	Libaec provides fast lossless compression of 1 up to 32 bit wide signed or unsigned integers (samples). The library achieves best results for low entropy data as often encountered in space imaging instrument data or numerical model output from weather or climate simulations. While floating point representations are not directly supported, they can also be efficiently coded by grouping exponents and mantissa.
libccc_user	libccc_user allows to access inside a job to information about execution time or memory consumed by the job
libcircle	libcircle is an API for distributing embarrassingly parallel workloads using self-stabilization.
libcudacompat	We define binary compatibility as a set of guarantees provided by the library, where an application targeting the said library will continue to work when dynamically linked against a different version of the library.
libflame/aocl	libflame is a portable library for dense matrix computations, providing much of the functionality present in LAPACK, developed by current and former members of the Science of High-Performance Computing (SHPC) group in the Institute for Computational Engineering and Sciences at The University of Texas at Austin. libflame includes a compatibility layer, lapack2flame, which includes a complete LAPACK implementation.
libgd	GD is an open source code library for the dynamic creation of images by programmers.
libint	a high-performance library for computing Gaussian integrals in quantum mechanics
libmpc	Mpc is a C library for the arithmetic of complex numbers with arbitrarily high precision and correct rounding of the result.
libmxml	Mini-XML is a small XML library that you can use to read and write XML and XML-like data files in your application without requiring large non-standard libraries
libnag	the NAG Library is a commercially available collection of numerical and statistical algorithms
libqglviewer	libQGLViewer is a C++ library based on Qt that eases the creation of OpenGL 3D viewers.
libquip	The QUIP package is a collection of software tools to carry out molecular dynamics simulations.
libtar	libtar is a library for manipulating tar files from within C programs.
libtree	libtree is a tool that turns ldd into a tree, explains why shared libraries are found and why not and optionally deploys executables and dependencies into a single directory
libvori	I have implemented our recently developed methods of Voronoi integration as well as the compressed bqb file format into a small C++ library, which is called libvori or Library for Voronoi Integration.
libxc	a library of exchange-correlation functionals for density-functional theory
libxsmm	LIBXSMM is a library for specialized dense and sparse matrix operations as well as for deep learning primitives such as small convolutions targeting Intel Architecture
loki	Loki is a C++ library of designs, containing flexible implementations of common design patterns and idioms.

Table 4 – continued from previous page

Name	Description
lwgrp	The light-weight group library defines data structures and collective operations to group MPI processes as an ordered set. Such groups are useful as substitutes for MPI communicators when the overhead of communicator creation is too costly.
magma	Matrix Algebra on GPU and Multicore Architectures
math/nvidia	Nvidia math libraries (like blas)
maven	Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.
med	The purpose of the MED module is to provide a standard for storing and recovering computer data associated to numerical meshes and fields, and to facilitate the exchange between codes and solvers.
mesquite	Mesh-Quality Improvement Library
metis	Serial Graph Partitioning and Fill-reducing Matrix Ordering
mkl	Intel Math Kernel Library
mpfr	The MPFR library is a C library for multiple-precision floating-point computations with correct rounding.
mumps	MUltifrontal Massively Parallel Sparse direct Solver
nccl_rdma_sharp_plugin	Optimized primitives for collective multi-GPU communication
netcdf-c	Network Common Data Form (C)
netcdf-cxx4	The netCDF-4 C++ API
netcdf-fortran	Network Common Data Form fortran support lib
netcdf95	Alternate Fortran interface to the NetCDF library
nlopt	Library for nonlinear local and global optimization
nvidia-index	NVIDIA IndeX is a 3D volumetric interactive visualization SDK
openblas	OpenBLAS is an optimized BLAS library based on GotoBLAS2 1.13 BSD version.
opencv	Visualization software tool
p3dfft	Scalable Framework for Three-Dimensional Fourier Transforms
parmetis	Parallel Graph Partitioning and Fill-reducing Matrix Ordering
parpack	Parallel ARPACK (PARPACK) and ARPACK. ARPACK is a collection of Fortran77 subroutines designed to solve large scale eigenvalue problems.
pastix	Parallel Sparse Matrix Package (sparse linear solver)
pcl	Portable Coroutine Library
pcre	The PCRE package contains Perl Compatible Regular Expression libraries.
petsc	Portable and extensible toolkit for scientific computation
plumed	An open source library for free energy calculations in molecular systems which works together with some of the most popular molecular dynamics engines
pmix	Process Management Interface (PMI) is a means of exchanging wireup information needed for interprocess communication. PMI Exascale is an extended version of the PMI standard specifically designed to support clusters up to and including exascale sizes.
pnetcdf	Parallel netCDF (PnetCDF) is an I/O library that supports data access to netCDF files in parallel
proj	Cartographic Projections library
ptscotch	Mesh generation library in Parallel
pytorch	An open source deep learning platform that provides a seamless path from research prototyping to production deployment.
rocr-runtime	The user-mode API interfaces and libraries necessary for host applications to launch compute kernels to available HSA ROCm kernel agents
roct-thunk-interface	The user-mode API interfaces used to interact with the ROCk driver

11.4. Libraries

Table 4 – continued from previous page

Name	Description
scalapack/aocl	AOCL Scalable Linear Algebra PACKage
scalapack/mkl	Intel Math Kernel Library ScaLAPACK routines
scalapack/netlib	Netlib Scalable Linear Algebra PACKage
scikit-learn	Simple and efficient tools for predictive data analysis
scotch	Mesh generation library
serf	High performance C-based HTTP client library built upon the Apache Portable Runtime (APR) library
sharp	Mellanox backend for hcoll
silo	Silo is a library for reading and writing a wide variety of scientific data to binary, disk files. The files Silo produces and the data within them can be easily shared and exchanged between wholly independently developed applications running on disparate computing platforms.
slepc	Scalable Library for Eigenvalue Problem computation
speos	
suitesparse	a suite of sparse matrix software
sundials	SUite of Nonlinear and DIfferential/ALgebraic equation Solvers
superlu	SuperLU is a general purpose library for the direct solution of large, sparse, non-symmetric systems of linear equations on high performance machines
superlu_dist	SuperLU is a general purpose library for the direct solution of large, sparse, non-symmetric systems of linear equations on high performance machines
tbb	Intel Threading Building Blocks Library
tensorflow	An open-source software library for Machine Intelligence
thrust	Thrust is a parallel algorithms library which resembles the C++ Standard Template Library (STL).
trilinos	The Trilinos Project provides algorithms within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems.
ucx	Mellanox backend for IB
voroplusplus	Voro++ is a software library for carrying out three-dimensional computations of the Voronoi tessellation.
wannier90	Wannier90 is an open-source code (released under GPLv2) for generating maximally-localized Wannier functions and using them to compute advanced electronic properties of materials with high efficiency and accuracy.
wi4pthread	Library developed by TGCC to solve threads placement issues
x264	x264 is a library and application for encoding video streams into the H.264/MPEG-4 AVC compression format
xerces-c	Xerces-C++ is a validating XML parser written in a portable
xz	compression software
yade	Yade is an extensible open-source framework for discrete numerical models, focused on Discrete Element Method.
yaml-cpp	yaml-cpp is a YAML parser and emitter in C++ matching the YAML 1.2 spec.
z3	Z3 is a theorem prover from Microsoft Research
zeromq	ZeroMQ looks like an embeddable networking library but acts like a concurrency framework.
zig	Zig is a general-purpose programming language and toolchain for maintaining robust, optimal and reusable software

11.5 Parallel

Table 5: Parallel products list

Name	Description
cuda	CUDA (Compute Unified Device Architecture) development tools for NVIDIA GPU
hcoll	Mellanox backend for MPI collectives
mpi/intelmpi	Intel MPI Library
mpi/openmpi	Open source MPI-2 implementation
mpi/wi4mpi	Allows to switch to any MPI implementation without recompiling
nccl	Optimized primitives for collective multi-GPU communication

11.6 Tools

Table 6: Tools products list

Name	Description
advisor	Threading design and prototyping tool for software architects
ant	Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other. The main known usage of Ant is the build of Java applications.
antlr	ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files
autoconf	Autoconf is an extensible package of M4 macros that produce shell scripts to automatically configure software source code packages.
automake	Automake is a tool for automatically generating Makefile.in files compliant with the GNU Coding Standards.
blender	Blender is the free and open source 3D creation suite.
cdo	CDO (Climate Data Operators) is a collection of command line Operators to manipulate and analyse Climate and NWP model Data.
cmake	Cross-platform, open-source build system
cppunit	Unit testing framework module for the C++ programming language, described as a C++ port of JUnit.
cubegui	cubegui, which is used as performance report explorer for Scalasca and Score-P
cubelib	cubelib, which is used as performance report explorer for Scalasca and Score-P
cubew	Cubew, which is used as performance report explorer for Scalasca and Score-P
cylc	Cylc is a workflow engine for cycling systems
darshan	Scalable HPC I/O characterization tool designed to capture an accurate picture of application I/O behavior, including properties such as patterns of access within files, with minimum overhead.
ddd	GNU DDD (Data Display Debugger), graphical front-end for GDB debugger
dmtcp	DMTCP (Distributed MultiThreaded Checkpointing) transparently checkpoints a single-host or distributed computation in user-space – with no modifications to user code or to the O/S.
doctest	doctest is a new C++ testing framework but is by far the fastest both in compile times (by orders of magnitude) and runtime compared to other feature-rich alternatives.
dotnet-sdk	.NET is the free, open-source, cross-platform framework for building modern apps and powerful cloud services

Table 6 – continued from previous page

Name	Description
doxygen	Doxygen is the de facto standard tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as C, Objective-C, C#, PHP, Java, Python, IDL (Corba and Microsoft flavors), Fortran, VHDL, Tcl, and to some extent D.
ffmpeg	FFmpeg is the leading multimedia framework, able to decode, encode, transcode, mux, demux, stream, filter and play pretty much anything that humans and machines have created.
flux-core	Core services for the Flux resource management framework
flux-sched	Core services for the Flux resource management framework
fypp	Fypp is a Python powered preprocessor. It can be used for any programming languages but its primary aim is to offer a Fortran preprocessor, which helps to extend Fortran with condititional compiling and template metaprogramming capabilities. Instead of introducing its own expression syntax, it uses Python expressions in its preprocessor directives, offering the consistency and versatility of Python when formulating metaprogramming tasks. It puts strong emphasis on robustness and on neat integration into developing toolchains.
gdb	A GNU source-level debugger for C, C++, Java and other languages
geany	Geany is a powerful, stable and lightweight programmer's text editor that provides tons of useful features without bogging down your workflow.
git	Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency
gitlab-runner	GitLab Runner is the open source project that is used to run your jobs and send the results back to GitLab. It is used in conjunction with GitLab CI, the open-source continuous integration service included with GitLab that coordinates the jobs.
glost	Greedly Launcher of Small Tasks
googletest	Google's framework for writing C++ tests on a variety of platforms
gprof	The GNU Profiler
gprof2dot	This is a Python script to convert the output from many profilers into a dot graph.
hpctoolkit	HPCToolkit is an integrated suite of tools for measurement and analysis of program performance on computers ranging from multicore desktop systems to the nation's largest supercomputers.
hwloc	The Portable Hardware Locality (hwloc) software package provides a portable abstraction (across OS, versions, architectures, . . .) of the hierarchical topology of modern architectures, including NUMA memory nodes, sockets, shared caches, cores and simultaneous multithreading. It also gathers various system attributes such as cache and memory information. It primarily aims at helping applications with gathering information about modern computing hardware so as to exploit it accordingly and efficiently.
igprof	the Ignominous Profiler
inspector	Memory and thread debugger
intltool	intltool is a set of tools to centralize translation of many different file formats using GNU gettext-compatible PO files.
ipm	IPM is a portable profiling infrastructure for parallel codes.
julia	high-level, high-performance dynamic programming language for technical computing
linaro-forge	C, C++ and F90 profiler and parallel Debugger for high performance and multi-threaded Linux applications (DDT, MAP. . .)
makedepf90	Makedepf90 is a program for automatic creation of Makefile-style dependency lists for Fortran source code.

Table 6 – continued from previous page

Name	Description
matlab	MATLAB is a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran.
medea	The MedeA software package is the leading environment for the atomistic simulation of materials
memonit	Memonit is a tool developed by TGCC staff in order to monitor the memory consumption of a job during time
mercurial	Mercurial is a free, distributed source control management tool. It efficiently handles projects of any size and offers an easy and intuitive interface
mmg	Mmg is an open source software for simplicial remeshing.
mono	Cross platform, open source .NET framework
mpifileutils	mpiFileUtils provides both a library called libmfu and a suite of MPI-based tools to manage large datasets, which may vary from large directory trees to large files.
mplayer	MPlayer is a movie player which runs on many systems
myqlm	myQLM is a quantum software stack for writing, simulating, optimizing, and executing quantum programs.
n2p2	If you have a working neural network potential setup (i.e. a settings file with network and symmetry function parameters, weight files and a scaling file) ready and want to predict energies and forces for a single structure you only need these components:
ncdu	Ncdu is a disk usage analyzer with an ncurses interface.
nco	The netCDF Operators (NCO) comprise command-line programs that take netCDF or HDF files as input, then operate and output the results to screen or files
nedit	NEdit is a fast, compact Motif/X11 plain text editor
ninja	A small build system with a focus on speed.
nsight-compute	NVIDIA Nsight Compute is an interactive kernel profiler for CUDA applications.
nsight-systems	NVIDIA Nsight Systems is a system-wide performance analysis tool designed to visualize an application's algorithm
nsight	Nvidia profiler
nvtop	NVTOP stands for Neat Videocard TOP, a (h)top like task monitor for GPUs and accelerators
octave	Octave is a high-level interpreted language, primarily intended for numerical computations
opari2	OPARI2, the successor of Forschungszentrum Juelich's OPARI,is a source-to-source instrumentation tool for OpenMP and hybrid codes. It surrounds OpenMP directives and runtime library calls with calls to the POMP2 measurement interface.
opencascade	Open CASCADE Technology (OCCT) is an object-oriented C++ class library designed for rapid production of sophisticated domain-specific CAD/CAM/CAE applications.
openstack	OpenStack is a set of software components that provide common services for cloud infrastructure.
otf2	The Open Trace Format 2 is a highly scalable, memory efficient event trace data format plus support library. It will become the new standard trace format for Scalasca, Vampir, and Tau and is open for other tools
pandoc	Pandoc is a Haskell library for converting from one markup format to another, and a command-line tool that uses this library
papi	Performance Application Programming Interface

Table 6 – continued from previous page

Name	Description
pdtoolkit	Program Database Toolkit (PDT) is a framework for analyzing source code written in several programming languages and for making rich program knowledge accessible to developers of static and dynamic analysis tools
perl	Perl 5 is a highly capable, feature-rich programming language with over 25 years of development.
pigz	Parallel implementation of gzip
pycharm	The Python IDE for Professional Developers
python3-extension	Python programming language
python3	Python programming language
r	R programming language
readline	The GNU readline library allows users to edit command lines as they are typed in. It can maintain a searchable history of previously entered commands, letting you easily recall, edit and re-enter past commands. It features both Emacs-like and vi-like keybindings, making its usage comfortable for anyone
reframe	ReFrame is a powerful framework for writing system regression tests and benchmarks, specifically targeted to HPC systems.
root	The ROOT system provides a set of OO frameworks with all the functionality needed to handle and analyze large amounts of data in a very efficient way.
scalasca	Software tool for measuring and analyzing performances of parallel programs
scons	Open Source software construction tool
scorep	the Scalable Performance Measurement Infrastructure for Parallel Codes is a software system that provides a measurement infrastructure for profiling, event trace recording, and online analysis of HPC applications
smokeview	Smokeview is a visualization program that displays the output of fds
sphinx	Create intelligent and beautiful documentation with ease
sqlite	SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine.
subversion	Subversion is a concurrent version control system
swift	OpenStack is a set of software components that provide common services for cloud infrastructure.
swig	SWIG is an interface compiler that connects programs written in C and C++ with scripting languages such as Perl, Python, Ruby, and Tcl.
sz	SZ is a modular parametrizable lossy compressor framework for scientific data (floating point and integers). It has applications in simulations, AI and instruments. It is a production quality software and a research platform for lossy compression. SZ is open and transparent. Open because all interested researchers and students can study or contribute to it. Transparent because all performance improvements are detailed in publications. SZ can be used for classic use-cases: visualization, accelerating I/O, reducing memory and storage footprint and more advanced use-cases like compression of DNN models and training sets, acceleration of computation, checkpoint/restart, reducing streaming intensity and running efficiently large problems that cannot fit in memory. Other use-cases will augment this list as users find new opportunities to benefit from lossy compression of scientific data
tau	TAU (Tuning and Analysis Utilities) is a portable profiling and tracing toolkit for performance analysis of parallel programs written in Fortran, C, C++, UPC, Java, Python
texlive	TeX Live is an easy way to get up and running with the TeX document production system.
themys	ParaView's branded application

continues on next page

Table 6 – continued from previous page

Name	Description
tkcvs	A Tcl/Tk based graphical interface to the CVS and Subversion configuration management systems. Support for Git and RCS too.
totalview	Dynamic source code and memory debugging for C, C++ and Fortran applications.
udunits	The UDUnits package from Unidata is a C-based package for the programatic handling of units of physical quantities.
uprof	AMD uProf is a performance analysis tool for applications. It allows developers to better understand the runtime performance of their application and to identify ways to improve its performance.
uranie	Uranie is a sensitivity and uncertainty analysis plateform based on the ROOT framework (http://root.cern.ch) . It is developed at CEA, the French Atomic Energy Commission (http://www.cea.fr).
valgrind	Debugging and memory checking tools
vampir	Profiling tool for parallel programs
vampirserver	Profiling tool for parallel programs
vdt	A collection of fast and inline implementations of mathematical functions
vef	VEF-Prospector is an open-source package for profiling MPI application in order to generate VEF trace-files. The VEF traces can be read by the VEF-TraceLIB to feed any network simulator.
vtune	Performance profiler for serial and parallel performance analysis
xcrysden	XCrySDen is a crystalline and molecular structure visualisation program aiming at display of isosurfaces and contours, which can be superimposed on crystalline structures and interactively rotated and manipulated.
xmlto	apply an XSL stylesheet to an XML document

JOB SUBMISSION

Computing nodes are shared between all the users of the computing center. A job scheduler manages the access to the global resources and allows users to *book* the resources they need for their computation. Job submission, resource allocation and job launch are handled by the batch scheduler called SLURM. An abstraction layer called Bridge is provided to give uniform command-line tools and more integrated ways to access batch scheduling systems.

To submit a batch job, you first have to write a shell script containing:

- a set of directives to tell which resources your job needs.

- instructions to execute your code.

You can launch the job by submitting its script to the batch scheduler. It will enter a batch queue. When resources become available, the job is launched over its allocated nodes. Jobs can be monitored.

12.1 Scheduling policy

Jobs are submitted to the supercomputer resource manager which will decide how to schedule them. There are hundreds of jobs submitted daily and the order in which they are launched depends on several parameters.

Jobs are sorted by priority. The priority is an integer value initially assigned to the job when it is submitted. The job with the highest priority will start first as soon as enough resources are available. Sometimes, a job with a lower priority may start first but only if it does not delay any job with a higher priority. This mechanism is called *backfill scheduling* and jobs can easily be backfilled if they require few computing resources and have a small time limit.

Here are the 3 components that are added to obtain the priority. They are listed below in order of importance:

- **The Quality of Service (QoS)**: The QoS component is a constant value associated to the QoS chosen during the submission. Its value depends on the priority factor given by the `ccc_mqinfo` command. This is the component that has the most influence on the overall value of the priority. Also, an usage limit can be set to prevent jobs from running as long as the project is not under a certain consumption. See *Qos* for more information.

- **The project's fair-share**: The fair-share component reflects the ratio between the total share of allocated hours and the amount of hours consumed for the chosen project. The fair-share value will be high if the project is under-consuming its allocated hours whereas it will be low if the project is over-consuming its allocated hours. A half-life decay is applied to the computation of the fair-share with a half life period of 14 days. That way, an over-consumption or under-consumption of hours will have a decreasing impact on new jobs submissions. After 2 months, the negative impact of an over-consumption is almost insignificant.

- **The job's age**: The age component depends on the time spent in a pending state while waiting for resources. Its value is incremented regularly for 7 days. After this delay, it will not increase anymore.

In order to reduce the waiting time as much as possible, try to:

- Use your computing hours evenly throughout your project duration. It will increase your fair-share value, thus increase the priority of your jobs.

- For small jobs, specify a time limit as close as possible to the real duration of the job instead of leaving default time limit of 2 hours. That way you are more likely to benefit from the *backfill scheduling* mechanism.

- Since the default time limit is 2 hours for all jobs, try to specify a time limit as close as possible to the real duration of your job. That way you are more likely to benefit from the *backfill scheduling* mechanism.

Note:

- Regarding the project's fair-share, the command *ccc_compuse* allows you to see the consumption status of your projects on each partition.

12.2 Choosing the file systems

Your job submissions can use the `-m` option to specify the file systems compute nodes will need for execution. This avoids job suspensions, if an unused file system becomes unavailable.

> example: `ccc_msub -m scratch,store` will run even if the WORK file system is unavailable. You can choose the following file systems: **scratch**, **work** and **store**.

> **Warning:** On Irene, your job submissions **MUST** use the `-m` option. Failing to provide the `-m` option will actually prevent your compute nodes from using any of **scratch**, **work** or **store**.

12.3 Submission scripts

12.3.1 ccc_msub

`ccc_msub <script>` is the command used to submit your batch job to the resource manager. The options passed to `ccc_msub` will determine the number of cores allocated to the job.

There are two ways of passing arguments to `ccc_msub`.

- Use the `#MSUB` keyword in the submission script. All the lines beginning with `#MSUB` will be parsed and the corresponding parameters will be taken into account by the batch manager.

- Use command line arguments. If the same argument is specified in the command line and the submission script, the command line argument will take precedence.

Note that you cannot pass arguments to the submission script when launching with `ccc_msub`.

For a short documentation and list of available options, see the help message:

```
$ ccc_msub -h
```

Basic options

- `-o output_file`: standard output file (special character `%I` will be replaced by the job ID)

- `-e error_file`: standard error file (special character `%I` will be replaced by the job ID)

All the output and error messages generated by the job are redirected to the log files specified with `-o` and `-e`. If you want all output to be redirected in one single file, just set the same name for `-o` and `-e`.

- `-r reqname`: job name

- `-A projid`: project/account name

- `-m filesystem`: file system required by the job. If another file system is unavailable, the job will not be impacted. Without the option, the job is considered to use every file system

  example: **ccc_msub -m scratch,store** will run even if the WORK file system is unavailable. You can choose the following file systems: **scratch**, **work** and **store**.

- `-@ mailopts`: mail options following the pattern `mailaddr[:begin|end|requeue]`

  example: **ccc_msub -@ jdoe@foo.com:begin,end** will send a mail to jdoe at the beginning and the end of the job default behavior depends of the underlying batch system

Partitions

The compute nodes are gathered in partitions according to their hardware characteristics (CPU architecture, amount of RAM, presence of GPU, etc). A partition is a set of identical nodes that can be targeted to host one or several jobs. Choosing the right partition for a job depends on code prerequisites in term of hardware resources. For example, executing a code designed to be GPU accelerated requires a partition with GPU nodes.

The partition is specified with the `-q` option. This option is mandatory while submitting.

The **ccc_mpinfo** command lists the available partitions for the current supercomputer. For each partition, it will display the number of available cores and nodes but also give some hardware specifications of the nodes composing the partition. For more information, see the *Supercomputer architecture* section.

Note:

- This choice is exclusive: your job can only be submitted on one of those architectures at a time.

- Job priority and limits are not related to the partition but to the QoS (Quality of Service) parameter (option `-Q`). QoS is the main component of job priority.

- Depending on the allocation granted to your project, you may not have access to all the partitions. You can check on which partition(s) your project has allocated hours thanks to the command **ccc_myproject**.

Resources

There are several options that will influence the number of cores that will be allocated for the job.

- `-n`: number of tasks that will be used in parallel mode (default=1)

- `-c`: number of cores per parallel task to allocate (default=1)

  If the `-c` parameter is not specified, the number of cores allocated is equal to the number of parallel tasks requested with `-n`. So specifying `-n` is enough for most of the basic jobs. The `-c` option is useful when each MPI process launched needs more resources than just one core. It can be either to require more cpu power or more memory:

- For hybrid MPI/OpenMP jobs, each MPI process will need several cores in order to spawn its OpenMP threads. In that case, usually, the -c option is equal to the number passed to OMP_NUM_THREADS

- For jobs requiring a lot of memory, each MPI process may need more than the 4 GB it is granted by default. In that case, the amount of memory may be multiplied by allocating more cores for each process. For example, by specifying -c 2, each process will be able to use the memory of 2 cores: 8GB.

- -N: number of nodes to allocate for parallel usage (default is chosen by the underlying system)

  The -N option is not necessary in most of the cases. The number of nodes to use is inferred by the number of cores requested and the number of cores per node of the partition used.

- -x: request for exclusive usage of allocated nodes

  The -x options forces the allocation of a whole node, even if only a couple of cores were requested. This is the default configuration for jobs requiring more than 128 cores.

- -T time_limit: maximum walltime of the batch job in seconds (optional directive, if not provided, set to default = 7200)

  It may be useful to set an accurate walltime in order to benefit from backfill scheduling.

Skylake partition

- To submit a job on a Skylake node, it's advised to recompile your code to use the Skylake supported vectorisation instructions as shown in the *Compiling for Skylake* section. Apart from the partition used, your submission script shouldn't change much :

```bash
#!/bin/bash
#MSUB -q skylake
#MSUB -n 40
ccc_mprun ./my-application
```

QoS

One can specify a Quality of Service (QoS) for each job submitted to the scheduler. The quality of service associated to a job will affect it in two ways: scheduling priority and limits. Depending on the required quantity of resources, on duration or on the job purpose (debugging, normal production, etc), you have to select the appropriate job QoS. It enables to trade a high job submission limit for a lower job priority, or a high job priority for lesser resources and duration.

ccc_mqinfo displays the available job QoS and the associated limitations:

```
$ ccc_mqinfo
Name     Partition  Priority  MaxCPUs  SumCPUs  MaxNodes  MaxRun  MaxSub    MaxTime
-------  ---------  --------  -------  -------  --------  ------  ------  ----------
long            *        20     2048     2048                        32  3-00:00:00
normal          *        20                                         128  1-00:00:00
test            *        40                          2               2    00:30:00
test    partition        40      280      280        10               2    00:30:00
```

For instance, to develop or debug your code, you may submit a job using the *test* QoS which will allow it to be scheduled faster. This QoS is limited to 2 jobs of maximum 30 minutes and 10 nodes each. CPU time accounting is not dependent on the chosen QoS.

To specify a QoS, you can use the -Q option of command-line or add #MSUB -Q <qosname> directive to your submission script. If no QoS is mentioned, default QoS *normal* will be used.

An usage limit can be set to manage hours consumption within a project. The usage limit can be set to *high*, *medium* or *low* with the option `#MSUB -U <limit>`.

- A job submitted with `-U high` priority will be scheduled normally as described in *Scheduling policy*. This is the default behavior if no usage limit is specified.

- A job submitted with `-U medium` will only be scheduled if the current project consumption is less than 20% over the *suggested use at this time*.

- A job submitted with `-U low` will only be scheduled if the current project consumption is more than 20% under the *suggested use at this time*.

Medium and low priority jobs may stay pending even if all the resources are free. It allows to prevent non important jobs from over consuming project hours and thus lower the default priority of future important jobs.

Dependencies between jobs

Note: The following methods are equivalent concerning the priorities of dependent jobs. See the *Scheduling policy* section for more details.

The command offers two options to prevent some jobs from running at the same time. Those options are `-w` and `-a`:

- The option `-w` makes jobs with the same submission name (specified with `-r`) run in the order they were submitted. For example, all jobs with the following options will not run at the same time.

```
#MSUB -r SAME_NAME
#MSUB -w
```

- The option `-a` indicates that a job depends on another (already submitted) job. Use it with the id of the existing job.

```
ccc_msub -a <existing_job> script.sh
```

- One can write multi step jobs that run sequentially by submitting the next job at the end of the current script. Here are some example scripts:

JOB_A.sh:

```
#!/bin/bash
#MSUB -r JOB_A
#MSUB -n 32
#MSUB -q <partition>
#MSUB -A <project>
ccc_mprun ./a.out
ccc_msub JOB_B.sh
```

JOB_B.sh:

```
#!/bin/bash
#MSUB -r JOB_B
#MSUB -n 16
#MSUB -q <partition>
#MSUB -A <project>
ccc_mprun ./b.out
ccc_msub JOB_C.sh
```

JOB_C.sh:

```
#!/bin/bash
#MSUB -r JOB_C
#MSUB -n 8
#MSUB -q <partition>
#MSUB -A <project>
ccc_mprun ./c.out
```

Then, only JOB_A.sh has to be submitted. When it finishes, the script submits JOB_B.sh, etc. . .

Note: If a job is killed or if it reaches its time limit, all the jobs are removed and the last **ccc_msub** may not be launched. To avoid this, you can use the **ccc_tremain** from libccc_user or use the #MSUB -w directive as described above.

Environment variables

When a job is submitted, some environment variables are set. Those variables may be used only within the job script.

- `BRIDGE_MSUB_JOBID`: batch id of the running job.

- `BRIDGE_MSUB_MAXTIME`: timelimit in seconds of the job.

- `BRIDGE_MSUB_PWD`: working directory. Usually the directory in which the job was submitted.

- `BRIDGE_MSUB_NPROC`: number of requested processes

- `BRIDGE_MSUB_NCORE`: number of requested cores per process

- `BRIDGE_MSUB_REQNAME`: job name

Note: You cannot use those variables as input arguments to **ccc_msub**. You cannot use them with the #MSUB headers.

12.3.2 ccc_mprun

ccc_mprun command allows to launch parallel jobs over nodes allocated by resources manager. So inside a submission script, a parallel code will be launched with the following command:

```
ccc_mprun ./a.out
```

By default, **ccc_mprun** takes information (number of nodes, number of processors, etc) from the resources manager to launch the job. You can customize its behavior with the command line options. Type **ccc_mprun -h** for an up-to-date and complete documentation.

Here are some basic options **ccc_mprun**:

- `-n nproc`: number of tasks to run

- `-c ncore`: number of cores per task

- `-N nnode`: number of nodes to use

- `-T time`: maximum walltime of the allocations in seconds (optional directive, if not provided, set to default=7200)

- `-E extra`: extra parameters to pass directly to the underlying resource manager

---

- `-m filesystem`: file system required by the job. If another file system is unavailable, the job will not be impacted. Without the option, the job is considered to use every file system. For example, **ccc_mprun -m scratch,store** will run even if the WORK file system is unavailable. You can choose the following file systems: **scratch**, **work** and **store**. This option may work depending the system and its configuration

---

Note: If the resources requested by **ccc_mprun** are not compatible with the resources previously allocated with **ccc_msub**, the job will crash with the following error message:

```
srun: error: Unable to create job step: More processors requested than permitted
```

12.4 Job submission scripts examples

- **Sequential job**

```bash
#!/bin/bash
#MSUB -r MyJob              # Job name
#MSUB -n 1                  # Number of tasks to use
#MSUB -T 600                # Elapsed time limit in seconds of the job (default: 7200)
#MSUB -o example_%I.o       # Standard output. %I is the job id
#MSUB -e example_%I.e       # Error output. %I is the job id
#MSUB -q <partition>        # Partition name (see ccc_mpinfo)
#MSUB -A <project>          # Project ID
set -x
cd ${BRIDGE_MSUB_PWD}       # The BRIDGE_MSUB_PWD environment variable contains the
→directory from which the script was submitted.
./a.out
```

- **Parallel MPI job**

```bash
#!/bin/bash
#MSUB -r MyJob_Para         # Job name
#MSUB -n 32                 # Number of tasks to use
#MSUB -T 1800               # Elapsed time limit in seconds
#MSUB -o example_%I.o       # Standard output. %I is the job id
#MSUB -e example_%I.e       # Error output. %I is the job id
#MSUB -q <partition>        # Partition name (see ccc_mpinfo)
#MSUB -A <project>          # Project ID
set -x
cd ${BRIDGE_MSUB_PWD}
ccc_mprun ./a.out
```

- **Parallel OpenMP/multi-threaded job**

```bash
#!/bin/bash
#MSUB -r MyJob_Para         # Job name
#MSUB -n 1                  # Number of tasks to use
#MSUB -c 16                 # Number of threads per task to use
#MSUB -T 1800               # Elapsed time limit in seconds
#MSUB -o example_%I.o       # Standard output. %I is the job id
#MSUB -e example_%I.e       # Error output. %I is the job id
```

(continues on next page)

<div align="right">(continued from previous page)</div>

```
#MSUB -q <partition>        # Partition name (see ccc_mpinfo)
#MSUB -A <project>          # Project ID
set -x
cd ${BRIDGE_MSUB_PWD}
export OMP_NUM_THREADS=16
 ./a.out
```

Note: An OpenMP/multi-threaded program can only run inside a node. If you ask more threads than available cores in a node, your submission will be rejected.

- **Parallel hybrid OpenMP/MPI or multi-threaded/MPI**

```
#!/bin/bash
#MSUB -r MyJob_ParaHyb            # Job name
#MSUB -n 8                        # Total number of tasks to use
#MSUB -c 4                        # Number of threads per task to use
#MSUB -T 1800                     # Elapsed time limit in seconds
#MSUB -o example_%I.o             # Standard output. %I is the job id
#MSUB -e example_%I.e             # Error output. %I is the job id
#MSUB -q <partition>              # Partition name
#MSUB -A <project>                # Project ID
set -x
cd ${BRIDGE_MSUB_PWD}
export OMP_NUM_THREADS=4
ccc_mprun ./a.out # This script will launch 8 MPI tasks. Each task will have 4 OpenMP␣
↪threads.
```

- **GPU jobs**

For examples of GPU job submission scripts, see *Running GPU applications*.

12.5 Job monitoring and control

Once a job is submitted with `ccc_msub`, it is possible to follow its evolution with several bridge commands.

We recommend you limit the rate at which your jobs query the batch system to an aggregate of 1 - 2 times / minutes. This includes all Bridge and Slurm queries such as `ccc_mpp`, `ccc_mstat`, `squeue`, `sacct`, or other Bridge and Slurm commands. Keep in mind this is an aggregate rate across all your jobs, so if you have a single job that queries once a minute but 500 of these jobs start at once the Slurm controller will see a rate of 500 queries / minute, so please scale your rate accordingly.

Warning: Use of `watch` for Slurm and Bridge commands is strongly prohibited.

12.5.1 ccc_mstat

The `ccc_mstat` command provides information about jobs on the supercomputer. By default, it displays all the jobs that are either running or pending on the different partitions of the supercomputer. Use the option -u to display only your jobs.

```
$ ccc_mstat -u
BATCHID  NAME    USER    PROJECT            QUEUE     QOS    PRIO   SUBHOST ␣
→EXEHOST   STA    TUSED    TLIM    MLIM   CLIM
-------  ----    ----    -------            -----     ------ ------ ------- -------
→   --- -------- -------- ------- ------
229862   MyJob1  mylogin projXXX@partition1  partition1 normal  210000 node174 ␣
→node174   PEN      0     86400    1865   2240
233463   MyJob2  mylogin projXXX@partition2  partition2 normal  210000 node175 ␣
→node1331  R00     58631   85680    1865   43200
233464   MyJob3  mylogin projXXX@partition3  partition3 normal  200000 node172 ␣
→node1067  R01     12171   85680    1865   43200
233582   MyJob4  mylogin projXXX@partition1  partition1 normal  200000 node172 ␣
→node1104  R01      3620   85680    1865   43200
```

Here is the information that can be gathered from :

- Basic job information (USER,PROJECT,BATCHID,CLIM,QUEUE,TLIM,NAME): Describes the parameter with which the job was submitted. It allows to check that the parameters passed to `ccc_msub` were taken into account correctly.

- PRIO: The priority of the job depends on many parameters. For instance, it depends on your project and the amount of hours your project consumed. It also depends on how long the job has been waiting in queue. The priority is what will determine the order in which the jobs from different users and different projects will run when the resource is available.

- SUBHOST: The host where the job was submitted.

- EXEHOST: The first host where the job is running.

- STA: The state of the job. Most of the time, it is either pending (PEN) if it is waiting for resources or running (R01) if it has started. Sometimes, the job is in a completing state (R00). It means the jobs has finished and computing resources are in a cleanup phase.

- TUSED/TLIM: If the job is running, the TUSED field shows for how long it has been running in seconds. The TLIM field shows the maximum execution time requested at submission.

- MLIM: The maximum memory allowed per core in MB.

- CLIM: The number of core requested at submission.

Here are command line options for `ccc_mstat`:

- `-f`: show jobs full name.

- `-q queue`: show jobs of requested batch queue.

- `-u [user]`: show jobs of a requested user. If no user is given, it shows the job of the current user.

- `-b batchid`: show all the processes related to a job.

- `-r batchid`: show detailed information of a running job.

- `-H batchid`: show detailed information of a finished job.

- `-O`: show jobs exceeding the time limit.

12.5.2 ccc_mpp

ccc_mpp provides information about jobs on the supercomputer. By default, it displays all the jobs that are either running or pending on the different partitions of the supercomputer. Use the option `-u $USER` to display only your jobs.

```
$ ccc_mpp -u $USER
USER    ACCOUNT BATCHID    NCPU    QUEUE       PRIORITY  STATE   RLIM    RUN/START  ␣
→SUSP   OLD     NAME    NODES/REASON
mylogin projXX  1680920    64      partition1  290281    RUN     1.0h    55.0s       - ␣
→   53.0s   MyJob1  node8002
mylogin projXX  1680923    84      partition2  284040    PEN     10.0h   -           - ␣
→   50.0s   MyJob3  Dependency
mylogin projXX  1661942    84      partition2  284040    RUN     24.0h   53.0s       - ␣
→   51.0s   MyJob3  node[1283-1285]
mylogin projXX  1680917    1024    partition2  274036    PEN     24.0h   -           - ␣
→   7.5m    MyJob4  Priority
mylogin projXX  1680921    28      partition3  215270    PEN     24.0h   ~05h36      - ␣
→   52.0s   MyJob2  Resources
```

Here is the information that can be gathered from :

- Basic job information (USER,ACCOUNT,BATCHID,NCPU,QUEUE,RLIM,NAME): Describes the parameter with which the job was submitted. It allows to check that the parameters passed to **ccc_msub** were taken into account correctly.

- PRIORITY: The priority of the job depends on many parameters. For instance, it depends on your project and the amount of hours your project consumed. It also depends on how long the job has been waiting in queue. The priority is what will determine the order in which the jobs from different users and different projects will run when the resource is available.

- STATE: The state of the job. Most of the time, it is either pending (PEN) if it is waiting for resources or running (RUN) if it has started. Sometimes, the job is in a completing state (COMP). It means the jobs has finished and computing resources are in a cleanup phase.

- RUN/START: If the job is running, this field shows for how long it has been running. If the job is pending, it sometimes gives an evaluation of the estimated start time. This start time may vary depending on the jobs submitted by other users.

- SUSP: The time spent in a suspended state. Jobs may be suspended by staff when an issue occur on the supercomputer. In that case, running jobs are not flushed but suspended in order to let them continue once the issue is solved.

- OLD: The total amount of time since the job was submitted. It includes the time spent waiting and running.

- NODES/REASON: If the job is running, this gives you the list of nodes used by the job. If it is pending, it gives you the reason. For example, "Dependency" means you submitted the job with a dependency to another unfinished job. "Priority" means there are other jobs that have a better priority and yours will start running after those. "Resources" means there are not enough cores available at the moment to let the job start. It will have to wait for some other jobs to end and free their allocated resources. "JobHeldAdmin" means that the current job has been held by a user who is not the owner (generaly an admin with the required rights). Please note that pending jobs with lower priority may display a pending reason not reflecting their current pending state since the batch scheduler only updates the pending reason of high priority jobs.

Here are command line options for **ccc_mpp**:

- `-r`: prints 'running' batch jobs

- `-s`: prints 'suspended' batch jobs

- `-p`: prints 'pending' batch jobs

- `-q queue`: requested batch queue

- `-u user`: requested user

- `-g group`: requested group

- `-n`: prints results without colors

12.5.3 ccc_mpeek

ccc_mpeek gives information about a job while it runs.

```
$ ccc_mpeek <jobid>
```

It is particularly useful to check the output of a job while it is running. The default behavior is to display the standard output. It is what you would basically find in the `.o` log file.

Here are command line options for **ccc_mpeek**:

- `-o`: prints the standard output

- `-e`: prints the standard error output

- `-s`: prints the job submission script

- `-t`: same as `-o` in **tail -f** mode

- `-S`: print the launched script of the running job

- `-d`: print the temporary directory of the running job

- `-I`: print the INFO file of the running job

12.5.4 ccc_mpstat

ccc_mpstat gives information about a parallel job during its execution.

```
$ ccc_mpstat <jobid>
```

It gives details about the MPI processes and their repartition across nodes with their rank and affinity.

Here are command line options for **ccc_mpstat**:

- `-r jobid` : display resource allocation characteristics

- `-a jobid` : display active steps belonging to a jobid

- `-t stepid` : print execution trace (tree format) for the specified stepid

- `-m stepid` : print mpi layout for the specified stepid

- `-p partition`: only print jobs of a particular partition

- `-u [user]` : only print jobs of a particular user or the current one if not specified

12.5.5 ccc_macct

Once the job is done, it will not appear in **ccc_mpp** anymore. To display information afterwards, the command **ccc_macct** is available. It works for pending and running jobs but information is more complete once the job has finished. It needs a valid jobID as input.

```
$ ccc_macct <jobid>
```

Here is an example of the output of the **ccc_macct** command:

```
$ ccc_macct 1679879


Jobid     : 1679879
Jobname   : MyJob
User      : login01
Account   : project01@partition
Limits    : time = 01:00:00 , memory/task = Unknown
Date      : submit = 14/05/2014 09:23:50 , start = 14/05/2014 09:23:50 , end = 14/05/
↪2014 09:30:00
Execution : partition = partition , QoS = normal , Comment = avgpowe+
Resources : ncpus = 32 , nnodes = 2
   Nodes=node[1802,1805]

Memory / step
-------------
              Resident Size (Mo)                    Virtual Size (Go)
JobID        Max      (Node:Task)     AveTask    Max   (Node:Task)          AveTask
-----------  ------------------------ -------    ----  ------------------------ -------
1679879.bat+   151 (node1802  :    0)     0     0.00 (node1802  :    0)    0.00
1679879.0      147 (node1805  :    2)     0     0.00 (node1805  :    2)    0.00
1679879.1      148 (node1805  :    8)     0     0.00 (node1805  :    8)    0.00
1679879.2        0 (node1805  :   16)     0     0.00 (node1805  :   16)    0.00


Accounting / step
-----------------


JobID         JobName          Ntasks  Ncpus Nnodes    Layout      Elapsed    Ratio
↪    CPusage    Eff  State
------------  ------------      ------  ----- ------    -------     -------    -----
↪    -------    ---  -----
1679879       MyJob                 -     32    2         -        00:06:10     100
↪         -      -  -
1679879.bat+  batch                 1      1    1      Unknown     00:06:10    100.0
↪         -      -  COMPLETED
1679879.0     exe0                  4      4    2       BBlock     00:03:18     53.5
↪   00:02:49   85.3  COMPLETED
1679879.1     exe1                 16     16    2       BBlock     00:02:42     43.7
↪   00:00:37   22.8  COMPLETED
1679879.2     exe2                 32     32    2       BBlock     00:00:03      .8
↪         -      -  FAILED


Energy usage / job (experimental)
---------------------------------
```

(continues on next page)

```
Nnodes  MaxPower  MaxPowerNode  AvgPower   Duration     Energy
------  --------  ------------  --------   --------     ------
     2      169W  node1802          164W   00:06:10     0kWh
```

Sometimes, there are several steps for the same job described in **ccc_macct**. Here, they are called *1679879.0*, *1679879.1* and *1679879.2*. This happens when there are several calls to **ccc_mprun** in one submission script. In this case, 3 executables were run with **ccc_mprun**: exe0, exe1 and exe2. Every other call to functions such as **cp**, **mkdir**, etc will be counted in the step called *1679879.batch*

There are 4 specific sections in the **ccc_macct** output.

- First section *Job summary*: It gives all the basic information about submission parameters you can also find in **ccc_mpp**. There is also submission date and time, when the job has started to run and when it has finished.

- Second section *Memory / step*: For each step, this section gives an idea of the amount of memory used by process. It gives the memory consumption of the top process.

- Third section *Accounting / step*: It gives detailed information for each step.

  - **Ntasks, Ncpus, Nnodes and Layout** describes how the job is distributed among the allocated resources. *Ntasks* is the number of processes defined by the parameter -n. By default, Ncpus is the same as Ntasks except if the number of cores per process was set with the -c parameter. The layout can be BBlock, CBlock, BCyclic or CCyclic depending on the process distribution. (more information given in the Advanced Usage documentation)

  - **Elapsed** is the user time spent in each step. **Ratio** is the percentage of time spent in this specific step compared to the total time of the job.

  - **CPusage** is the average cpu time for all the processes in one step and **Eff** is defined by *CPusage/Elapsed*100*. Which means that if Eff is close to 1, all the processes are equally busy.

- Fourth section *Energy usage*: It gives an idea of the amount of electric energy used while running this job. This section is only relevant if the job used entire nodes.

12.5.6 ccc_mdel

ccc_mdel enables to kill your jobs. It works whether they are running or pending.

First, identify the batchid of the job you need to kill with **ccc_mpp** for example. And then, kill the job with:

```
$ ccc_mdel <jobid>
```

12.5.7 ccc_malter

ccc_malter enables to decrease your jobs time limit. It works only when they are running or pending.

```
$ ccc_malter -T <new time limit> <jobid>
```

Here are available options:

- -T time_limit: Decrease the time limit of a job

- -L licenses_string: Change licenses of your job

12.5.8 ccc_affinity

The command **ccc_affinity** show you the processes and threads affinity for a given job id. The usual format is
ccc_affinity [options] JOBID.

Here are available options:

- -l: Run on local node.

- -t: Display processes and threads. Default is to display processes only.

- -u: Specify a username.

This is an example of output:

```
$ ccc_affinity 900481
Host            Rank  PID       %CPU  State MEM_kB    CPU   AFFINITY              NAME
node1434:
|               2     8117      310   Sl    34656     28    0,28                  life_
↪par_step7
|               3     8118      323   Rl    34576     42    14,42                 life_
↪par_step7
node1038:
|               0     6518      323   Rl    34636     0     0,28                  life_
↪par_step7
|               1     6519      350   Rl    34732     42    14,42                 life_
↪par_step7
```

And this is with thread option -t:

```
$ ccc_affinity -t 900481
Host            Rank  PID       %CPU   State ThreadID   MEM_kB    CPU   AFFINITY      ␣
↪      NAME
node1434:
|               2     8117      20.3   Sl    8117       34660     28    0,28          ␣
↪      life_par_step7
|               `--   --        0.0    Sl    8125       34660     29    0-13,28-41    ␣
↪      life_par_step7
|               `--   --        0.0    Sl    8126       34660     0     0-13,28-41    ␣
↪      life_par_step7
|               `--   --        0.0    Sl    8142       34660     29    0-13,28-41    ␣
↪      life_par_step7
|               `--   --        0.0    Sl    8149       34660     36    0-13,28-41    ␣
↪      life_par_step7
|               `--   --        99.6   Rl    8150       34660     4     4,32          ␣
↪      life_par_step7
|               `--   --        99.6   Rl    8151       34660     7     7,35          ␣
↪      life_par_step7
|               `--   --        99.6   Rl    8152       34660     11    11,39         ␣
↪      life_par_step7
|               3     8118      33.6   Rl    8118       34580     42    14,42         ␣
↪      life_par_step7
|               `--   --        0.0    Sl    8124       34580     43    14-27,42-55   ␣
↪      life_par_step7
|               `--   --        0.0    Sl    8127       34580     18    14-27,42-55   ␣
↪      life_par_step7
```

```
|               `--   --        0.0    Sl   8143     34580     50    14-27,42-55  ␣
↪        life_par_step7
|               `--   --        0.0    Sl   8145     34580     23    14-27,42-55  ␣
↪        life_par_step7
|               `--   --       99.6    Rl   8146     34580     18    18,46        ␣
↪        life_par_step7
|               `--   --       99.6    Rl   8147     34580     21    21,49        ␣
↪        life_par_step7
|               `--   --       99.6    Rl   8148     34580     25    25,53        ␣
↪        life_par_step7
node1038:
|          0    6518          44.1    Rl   6518     34636     28    0,28         ␣
↪        life_par_step7
|               `--   --        0.0    Sl   6526     34636     29    0-13,28-41   ␣
↪        life_par_step7
|               `--   --        0.0    Sl   6531     34636     11    0-13,28-41   ␣
↪        life_par_step7
|               `--   --        0.0    Sl   6549     34636     36    0-13,28-41   ␣
↪        life_par_step7
|               `--   --        0.0    Sl   6553     34636     10    0-13,28-41   ␣
↪        life_par_step7
|               `--   --       99.8    Rl   6554     34636      4    4,32         ␣
↪        life_par_step7
|               `--   --       99.8    Rl   6555     34636      7    7,35         ␣
↪        life_par_step7
|               `--   --       99.8    Rl   6556     34636     11    11,39        ␣
↪        life_par_step7
|          1    6519          71.1    Sl   6519     34736     42    14,42        ␣
↪        life_par_step7
|               `--   --        0.0    Sl   6525     34736     43    14-27,42-55  ␣
↪        life_par_step7
|               `--   --        0.0    Sl   6527     34736     17    14-27,42-55  ␣
↪        life_par_step7
|               `--   --        0.0    Sl   6548     34736     43    14-27,42-55  ␣
↪        life_par_step7
|               `--   --        0.0    Sl   6557     34736     50    14-27,42-55  ␣
↪        life_par_step7
|               `--   --       99.6    Rl   6558     34736     18    18,46        ␣
↪        life_par_step7
|               `--   --       99.6    Rl   6559     34736     21    21,49        ␣
↪        life_par_step7
|               `--   --       99.6    Rl   6560     34736     25    25,53        ␣
↪        life_par_step7
```

We can see which process runs on which node and which thread runs on which core/CPU. The "State" column show if the thread is Running (R) or sleeping (S). The AFFINITY column shows CPUs on which a thread is allowed to move on and the CPU column shows the CPU on which a thread is currently running.

On a single node we can see CPU numbered to 55 though a node is 28 cores. It's because of Intel Hyper-Threading that allows to run two threads on one core.

12.6 Special jobs

Note: A step (or SLURM step) is a call of **ccc_mprun** command inside an allocation or a job.

Note: In the following cases, some scripts use the submission directive **#MSUB -F**. This directive call a Bridge plugin which uses Flux. It is a new framework for managing resources and jobs.

> **Warning:** If you want to submit a chained job at the end of a job using the Flux plugin, please write the following command before the submit command:

```
BATCH_SYSTEM=slurm ccc_msub [...]
```

12.6.1 Multiple sequential short steps

In this case, a job contains many sequential and very short steps.

```bash
#!/bin/bash
#MSUB -r MyJob_SeqShort          # Job name
#MSUB -n 16                      # Number of tasks to use
#MSUB -T 3600                    # Elapsed time limit in seconds
#MSUB -o example_%I.o            # Standard output. %I is the job id
#MSUB -e example_%I.e            # Error output. %I is the job id
#MSUB -q <partition>             # Partition name
#MSUB -A <project>               # Project ID
#MSUB -F                         # Use Flux plugin


for i in $(seq 1 10000)
do
   ccc_mprun -n 16 ./bin1
done
```

12.6.2 Embarrassingly parallel jobs

An embarrassingly parallel job is a job which launches independent processes in parallel. These processes need few or no communications. We call such an independent process a task.

Multiple concurrent sequential short steps

In this case, a job contains many concurrent, sequential and very short steps.

```bash
#!/bin/bash
#MSUB -r MyJob_MulSeqShort          # Job name
#MSUB -n 16                         # Number of tasks to use
#MSUB -T 3600                       # Elapsed time limit in seconds
#MSUB -o example_%I.o               # Standard output. %I is the job id
#MSUB -e example_%I.e               # Error output. %I is the job id
#MSUB -q <partition>                # Partition name
#MSUB -A <project>                  # Project ID
#MSUB -F                            # Use Flux plugin


for i in $(seq 1 10000)
do
   ccc_mprun -n 8 ./bin1 &
   ccc_mprun -n 8 ./bin2 &
   wait
done
```

Multiple concurrent steps

This case is the typical embarrassingly job: a job allocates resources and launches multiple steps which are independant from other steps. Flux will launch them through the available resources. When a step ends, a new step will be launched if there are enough resources.

The different tasks to launch must be listed in a simple text file. Each line of this taskfile contains:

- the number of task
- the number of core per task
- the command to launch

For example:

```
$ cat taskfile.txt
8-2 bin1.exe
1-1 bin.exe file1.dat
2-8 bin2.exe file2.dat
<...>
4-1 bin2.exe file2.dat
```

And the job script:

```bash
#!/bin/bash
#MSUB -r MyJob_Concurrent           # Job name
#MSUB -n 256                        # Number of tasks to use
```

```
#MSUB -T 3600                    # Elapsed time limit in seconds
#MSUB -o example_%I.o            # Standard output. %I is the job id
#MSUB -e example_%I.e            # Error output. %I is the job id
#MSUB -q <partition>             # Partition name
#MSUB -A <project>               # Project ID
#MSUB -F                         # Use Flux plugin


ccc_mprun -B taskfile.txt
```

You can combine the previous actions into one:

```
#!/bin/bash
#MSUB -r MyJob_Concurrent        # Job name
#MSUB -n 256                     # Number of tasks to use
#MSUB -T 3600                    # Elapsed time limit in seconds
#MSUB -o example_%I.o            # Standard output. %I is the job id
#MSUB -e example_%I.e            # Error output. %I is the job id
#MSUB -q <partition>             # Partition name
#MSUB -A <project>               # Project ID
#MSUB -F                         # Use Flux plugin


ccc_mprun -B <(cat << EOF
8-2 bin1.exe
1-1 bin.exe file1.dat
2-8 bin2.exe file2.dat
<...>
4-1 bin2.exe file2.dat
EOF
)
```

Multiple concurrent jobs

In this cas, you can submit many jobs inside an allocation. Bridge provides an environment variable
`BRIDGE_MSUB_ARRAY_TASK_ID` to differentiate the different subjobs.

For example, here a job script where you need you submit 10000 times:

```
$ cat subjob.sh
#!/bin/bash
<some preprocessing commands>
ccc_mprun bin.exe file_${BRIDGE_MSUB_ARRAY_TASK_ID}.dat
<some postprocessing commands>
```

And the job script:

```
#!/bin/bash
#MSUB -r MyJob_ConcurrentJobs    # Job name
#MSUB -n 256                     # Number of tasks to use
#MSUB -T 3600                    # Elapsed time limit in seconds
#MSUB -o example_%I.o            # Standard output. %I is the job id
#MSUB -e example_%I.e            # Error output. %I is the job id
#MSUB -q <partition>             # Partition name
```

```
#MSUB -A <project>                # Project ID
#MSUB -F                          # Use Flux plugin


flux resource drain 0
ccc_msub -n 16 -y 1-10000 subjob.sh
flux resource undrain 0
```

In this example, the job will allocate 256 tasks. Inside, it will submit 10000 subjobs.

GLoST: Greedy Launcher of Small Tasks

GLoST is a lightweight, highly scalable tool for launching independent non-MPI processes in parallel. It has been developed by the TGCC for handling huge to-do list like operations. The source code is available on the cea-hpc github. GLoST manages the launch and scheduling of a list of tasks with the command **glost_launch**. It also allows error detection, continuation of undone operations and relaunch of failed operations thanks to the post processing script `glost_filter.sh`.

The different tasks to launch must be listed in a simple text file. Commented and blank lines and supported. Comments added at the end of a line will be printed to the job output. This can be used to tag different tasks. Here is an example of a task file:

```
$ cat taskfile.list
./bin1 # Tag for task 1
./bin2 # Tag for task 2
./bin3 # Tag for task 3
./bin4 # Tag for task 4
./bin5 # Tag for task 5
./bin6 # Tag for task 6
./bin7 # Tag for task 7
./bin8 # Tag for task 8
./bin9 # Tag for task 9
./bin10 # Tag for task 10
```

Or with MPI binaries:

```
$ cat task_mpi.list
ccc_mprun -E"--jobid=${SLURM_JOBID}" -E"--exclusive" -n 3 ./mpi_init
ccc_mprun -E"--jobid=${SLURM_JOBID}" -E"--exclusive" -n 5 ./mpi_init
```

Here is an example of submission script:

```
#!/bin/bash
#MSUB -r MyJob_Para               # Job name
#MSUB -n 4                        # Number of tasks to use
#MSUB -T 3600                     # Elapsed time limit in seconds
#MSUB -o example_%I.o             # Standard output. %I is the job id
#MSUB -e example_%I.e             # Error output. %I is the job id
#MSUB -q <partition>              # Partition name
#MSUB -A <project>                # Project ID


module load glost
ccc_mprun glost_launch taskfile.list
```

GLoST will automatically manage the different tasks and schedule them on the different available nodes. Note that one MPI process is reserved for task management so GLoST cannot run on 1 process. For more information, please check out the man page for `glost_launch`.

Once the job is submitted, information on the task scheduling is provided in the error output of the job. For each task, it shows which process launched it, its exit status and its duration. Here is a typical output for a 4 process job. Process 0 is used for scheduling and the 10 tasks to launch are treated by processes 1,2 or 3.

```
#executed by process    3 in 7.01134s     with status    0 : ./bin3 # Tag for task 3
#executed by process    1 in 8.01076s     with status    0 : ./bin2 # Tag for task 2
#executed by process    2 in 9.01171s     with status    0 : ./bin1 # Tag for task 1
#executed by process    2 in 0.00851917s  with status    12 : ./bin6 # Tag for task 6
#executed by process    2 in 3.00956s     with status    0 : ./bin7 # Tag for task 7
#executed by process    1 in 5.01s        with status    0 : ./bin5 # Tag for task 5
#executed by process    3 in 6.01114s     with status    0 : ./bin4 # Tag for task 4
```

Some tasks may exit on errors or not be executed if the job has reached its timelimit before launching all the tasks in the taskfile. To help analysing the executed, failed or not executed tasks, we provide the script `glost_filter.sh`. By default, lists all the executed tasks and adding the option `-H` will highlight the failed tasks. Let's say the error output file is called `example_1050311.e`. Here are some useful options:

- List all failed tasks (with exit status other than 0)

```
$ glost_filter.sh -n taskfile example_1050311.e
./bin6 # Tag for task 6
```

- List all tasks not executed (may be used to generate the next taskfile)

```
$ glost_filter.sh -R taskfile example_1050311.e
./bin8 # Tag for task 8
./bin9 # Tag for task 9
./bin10 # Tag for task 10
```

For more information and options, please check out `glost_filter.sh -h`.

GLoST also provides the tool `glost_bcast.sh` which broadcasts a file from a shared filesystem to the local temporary directory (`/tmp` by default). Typical usage on the center is:

```
#!/bin/bash
#MSUB -r MyJob_Para              # Job name
#MSUB -n 32                      # Number of tasks to use
#MSUB -T 3600                    # Elapsed time limit in seconds
#MSUB -o example_%I.o            # Standard output. %I is the job id
#MSUB -e example_%I.e            # Error output. %I is the job id
#MSUB -q <partition>            # Partition name
#MSUB -A <project>              # Project ID


module load glost

# Broadcast <file> on each node under /tmp/<file>
ccc_mprun -n ${BRIDGE_MSUB_NNODE} -N ${BRIDGE_MSUB_NNODE} glost_bcast <file>

# Alternatively, to broadcast <file> on each node under /dev/shm/<file>
TMPDIR=/dev/shm ccc_mprun -n ${BRIDGE_MSUB_NNODE} -N ${BRIDGE_MSUB_NNODE} glost_bcast
→<file>
```

12.6.3 MPMD jobs

An MPMD job (for Multi Program Multi Data) is a parallel job that launches different executables over the processes. The different codes are still sharing the same MPI environment. This can be done with the -f option of **ccc_mprun** and by creating an appfile. The appfile should specify the different executables to launch and the number of processes for each.

Homogeneous

An homogeneous MPMD job is a parallel job where each process have the same cores number. Here is an example of an appfile:

```
$ cat app.conf
1       ./bin1
5       ./bin2
26      ./bin3

# This script will launch the 3 executables
# respectively on 1, 5 and 26 processes
```

And the submission script:

```
#!/bin/bash
#MSUB -r MyJob_Para                 # Job name
#MSUB -n 32                         # Total number of tasks/processes to use
#MSUB -T 1800                       # Elapsed time limit in seconds
#MSUB -o example_%I.o               # Standard output. %I is the job id
#MSUB -e example_%I.e               # Error output. %I is the job id
#MSUB -q <partition>                # Partition name
#MSUB -A <project>                  # Project ID
set -x
cd ${BRIDGE_MSUB_PWD}

ccc_mprun -f app.conf
```

Note: The total number of processes specified on the appfile cannot be larger than the number of processes requested for the job in the submission script.

In order to have several calls per line in the appfile, it is necessary to execute the whole line in the **bash** command.

- For example, if each binary has to be executed in its own directory

```
1 bash -c "cd ./dir-1 && ./bin1"
4 bash -c "cd ./dir-2 && ./bin2"
```

- Or if you need to export an environment variable that is different for each binary.

```
1 bash -c "export OMP_NUM_THREADS=3; ./bin1;"
4 bash -c "export OMP_NUM_THREADS=1; ./bin2;"
```

Heterogeneous

An heterogeneous MPMD job is a parallel job where each process could have a different threads number. Heterogeneous MPMD is enabled by loading **feature/bridge/heterogenous_mpmd** module. Here is an example of an appfile:

```
$ cat app.conf
1-2 bash -c "export OMP_NUM_THREADS=2; ./bin1"
5-4 bash -c "export OMP_NUM_THREADS=4; ./bin2"
2-5 bash -c "export OMP_NUM_THREADS=5; ./bin3"

# This script will launch the 3 executables
# respectively on 1, 5 and 2 processes with 2, 4 and 5 cores
# 1*2 + 5*4 + 2*5 = 32 (#MSUB -n 32)
```

The first number describes how many processes to run the followed command while the second one indicates the number of cores allocated to each process.

And the submission script:

```
#!/bin/bash
#MSUB -r MyJob_Para              # Job name
#MSUB -n 32                      # Total number of tasks and cores to use
#MSUB -T 1800                    # Elapsed time limit in seconds
#MSUB -o example_%I.o            # Standard output. %I is the job id
#MSUB -e example_%I.e            # Error output. %I is the job id
#MSUB -q <partition>             # Partition name
#MSUB -A <project>               # Project ID
set -x
cd ${BRIDGE_MSUB_PWD}
module load feature/bridge/heterogenous_mpmd

ccc_mprun -f app.conf
```

PROJECT ACCOUNTING

13.1 Computing hours consumption control process

To guarantee a smooth and fair use of the computing resources available on Irene, projects must use their awarded core hours on a regular basis without being late.

To encourage users to respect this rule, projects consumption is automatically and periodically checked. In case of under-consumption, the number of core hours awarded gets reduced. On the other hand, this mechanism doesn't prevent over-consumption.

The process respects the following rules:

- The 15th of each month, the current accounting of each project is compared with the theoretical one (awarded hours / total number of days in project * elapsed number of days).

- Between 30 days and 59 days of lateness, an e-mail warns the project leader.

- If a project has more than 60 days of lateness, a 30-days amount of core hours is deducted from the global awarded core hours.

Note:

- This process concerns PRACE Regular Access projects and over-a-million-hour GENCI projects.

- At any time, users can check their project accounting with the "ccc_myproject" command.

- No hour will be subtracted without a warning e-mail being sent a month beforehand.

- Derogation requests must be sent to the TGCC Hotline.

In case of over-consumption, jobs still can be run. They will run only if there is no job from a nominal-consuming or from an under-consuming project in queue waiting for computing resources. A running job from an over-consuming project is called a "bonus job". This mechanism is automatically managed by the scheduler and no option is necessary to use it. For DARI projects, the project is closed when the consumed hours reach 125% of the granted hours amount and can therefore consume more than the initially granted hours amount if it has benefited of bonus jobs.

13.2 ccc_myproject

It is very important to consume the project awarded hours or your computing share on a regular basis. Therefore, you should check the project consumption. The command **ccc_myproject** gives information about the accounting of your project(s)

Precisely, it shows you project consumption details for each partition as well as **expected** consumption: it helps users to have "linear" consumption for the duration of the project and avoid consumption peaks. It is recommended to use the hours allocated to a project on a regular basis.

ccc_myproject may indicate a delay in the expected average consumption ("real use" < "suggested use"). However, the longer you wait for jobs in the machine, the more you have consumed in the last few days: this over- or under-consumption status of your project is given by the command **ccc_compuse**.

13.2.1 For PRACE and GENCI projects

```
$ ccc_myproject
Accounting for project XXXXXXX on Machine at 2014-04-13
Login                           Time in hours
login01     ...........................75382.44
login02     .............................20.02


Total       ..........................75402.46
Allocated   .......................2000000.00


Suggested use at this time...............12.25%
Real use at this time     ................3.75%


Project deadline 201X-0X-0X
```

You will find:

- the consumed compute time in hours for each member of the project

- the total consumed compute time of the project

- the total amount of hours allocated for this project

- the project's deadline

The suggested use gives the percentage of hours you should ideally have used at this point depending on your total amount of allocated hours and your project deadline. Your real use should be as close as possible to the suggested value.

If you are part of several projects, information will be displayed for each project you are member of.

Accounting information are updated once a day.

13.2.2 For CCRT partners

```
$ ccc_myproject

Accounting for partner XXXXXXX on partition1 at 2012-11-07
Login        Imputation     Hours (1 month)     Hours (3 months)     Hours (1 year)

login01      *****         ................0.00................19.10................19.10
login02      *****         ..............609.33.............1187.34.............1187.34
login03      *****         ..............472.89..............472.89..............472.89
.....


Total used                 ............38397.85............59700.49............59700.49
Total suggested used  ...........226271.24...........671514.63..........2671460.39
```

This information is given in the following periods: 30 days rolling, 3 months rolling, 1 year rolling.

The suggested use gives the amount of hours you should ideally have used at this point depending on your total amount corresponding to your computing share. Your real use should be as close as possible to the suggested value.

Accounting information are updated once a day.

13.3 ccc_compuse

The command ccc_compuse indicates how the batch scheduler prioritize of your next job(s) according to your project(s) and its associated partition.

```
$ ccc_compuse
Account Status
-------------------------------------------------
genxxxx@broadwell regular consumption
genxxxx@skylake under-consumption
```

It as the following 3 states:

- under-consumption: you will get a higher priority

- regular consumption: you will get a regular priority

- over-consumption: your will get a decrease priority

All your project usage is taken into account but with digressive weighting factors from 1 to 0. For your information a job which ran 14 days ago gets a 0.5 weighting factor.

Moreover the over-consumption is not directly computed from a ratio between the attributed hours and the consumed hours. In fact, it is computed with a comparison between, on one side the percentage of the consumed hours of your project to the whole available hours of the partition, and on the other side the percentage of the attributed hours of your project on the partition. Usually the 2 methods are closely equivalent, but in special cases such as during a Grand Challenge where there are only several projects and more unavailability, their results may differ significantly.

FOURTEEN

COMPILATION

14.1 Language standard

Most programming languages are defined through norms or standards. For example the C language have several standards:

- K&R C
- ANSI C (C89) – ISO C (C90)
- C99
- C11
- C embedded

These standards define the syntax of the language, the meaning of these syntactic constructs regarding the resulting machine code. Most of them are a revision of the previous standard, incorprorating changes and new features.

The Intel Suite compilers, as well as the GNU Compiler Collection supports most releases the C, C++ and Fortran languages. However, some compilation options and optimisation as well as support for specific features of the most recent iterations of these standards are compiler-specific. Please refer to their respective documentations for more information.

14.2 Available compilers

Several well-known compilers are available for the C, C++ and Fortran languages. The most common being:

- Intel Compiler suite (`icc`, `icpc`, `ifort`)
- GNU compiler suite (`gcc`, `g++`, `gfortran`)

To get a complete list of available compilers and versions use the **search** parameter of **module**

```
$ module search compiler
```

We recommend to use the Intel Compiler Suite for better performances.

Here is how you would basically compile a serial code

```
$ icc [options] -o serial_prog.exe serial_prog.c
$ icpc [options] -o serial_prog.exe serial_prog.cpp
$ ifort [options] -o serial_prog.exe serial_prog.f90
```

14.2.1 Intel

Compiler flags

The following sections are an overview of the most frequent options for each compiler.

C/C++

The Intel compilers `icc` and `icpc` use mostly the same options. Their behaviors differ slightly: `icpc` assumes that all source files are C++, whereas `icc` distinguishes between .c and .cpp filenames.

Basic flags:

- `-o exe_file`: names the executable `exe_file`
- `-c`: generates the corresponding object file, without creating an executable.
- `-g`: compiles with the debug symbols.
- `-I dir_name`: specifies the path of the include files.
- `-L dir_name`: specifies the path of the libraries.
- `-l bib`: asks to link the `libbib.a` library

Preprocessor:

- `-E`: preprocess the files and sends the result to the standard output
- `-P`: preprocess the files and sends the result in `file.i`
- `-Dname=`: defines the `name` variable
- `-M`: creates a list of dependencies

Practical:

- `-p`: profiling with gprof (needed at compilation time)
- `-mp`, `-mp1`: IEEE arithmetic, mp1 is a compromise between time and accuracy

To tell the compiler to conform to a specific language standard :

- `-std=val`: where **val** can take the following values (cf. **man icc** or **man icpc**)
    - c++14 : Enables support for the 2014 ISO C++ standard features.
    - c++11 : Enables support for many C++11 (formerly known as C++0x) features.
    - C99 : Conforms to The ISO/IEC 9899:1999 International Standard for the C language.
    - …

Note: If the desired standard is not fully supported by the current version of the Intel compiler (some features of the standard are not yet implemented), it might be supported by the last version of the GNU compilers. Since the Intel suit compilers are compiled against the gcc system (the one from the OS), load a recent version of the GNU compilers might solve this issue.

Fortran

The Intel compiler for Fortran is `ifort`.

Basic flags:

- `-o` *exe_file*: name the executable `exe_file`
- `-c`: generate the object file without creating an executable.
- `-g`: compile with the debug symbols.
- `-I dir_name`: add `dir_name` to the list of directories where include files are looked for.
- `-L dir_name`: add `dir_name` to the list of directories where libraries are looked for.
- `-l bib`: link the `libbib.a` library

Run-time check

- `-C` or `-check`: generates a code which ends up in 'run time error' (ex: segmentation fault)

Preprocessor:

- `-E`: pre-process the files and send the result to the standard output
- `-P`: pre-process the files and send the result to `file.i`
- `-Dname=`: assign the value *value* to the variable `name`
- `-M`: creates a list of dependencies
- `-fpp`: pre-process the files and compile

Practical:

- `-p`: compile for profiling with **gprof**. You will not be able to use **gprof** otherwise.
- `-mp`, `-mp1`: IEEE arithmetic (mp1 is a compromise between time and accuracy)
- `-i8`: promote integers to 64 bits by default
- `-r8`: promote reals to 64 bits by default
- `-module dir`: send/read the files `*.mod` in the `dir` directory
- `-fp-model strict`: strictly adhere to value-safe optimizations when implementing floating-point calculations, and enable floating-point exception semantics. This may slow down your program.

To tell the compiler to conform to a specific language standard :

- `-stand=val`: where **val** can take the following values (cf. **man ifort**)
  - f15: Issues messages for language elements that are not standard in draft Fortran 2015.
  - f08: Tells the compiler to issue messages for language elements that are not standard in Fortran 2008
  - f03: Tells the compiler to issue messages for language elements that are not standard in Fortran 2003
  - …

Note: Please refer to the *man pages* for more information about the compilers.

Optimization flags

Compilers provide many optimization options: this section describes them.

Basic optimization options :

- `-O0`, `-O1`, `-O2`, `-O3`: optimization levels - default: `-O2`

- `-opt_report` : writes an optimization report to stderr (`-O3` required)

- `-ip`, `-ipo`: inter-procedural optimizations (mono and multi files). The command xiar must be used instead of ar to generate a static library file with objects compiled with `-ipo` option.

- `-fast` : default high optimization level (`-O3 -ipo -static`).

- `-ftz` : considers all the denormalized numbers (like INF or NAN) as zeros at runtime.

- `-fp-relaxed` : mathematical optimization functions. Leads to a small loss of accuracy.

- `-pad` : makes the modification of the memory positions operational (**ifort** only)

> **Warning:** The `-fast` option is not allowed with MPI because the MPI context needs some libraries which only exist in dynamic mode. This is incompatible with the `-static` option. You need to replace `-fast` by `-O3 -ipo`.

Vectorization flags

Some options allow to use specific vectorization instructions of Intel processors to optimize the code. They are compatible with most Intel processors. The compiler will try to generate these instructions if the processor allows them.

- `-xcode` : Tells the compiler which processor features it may target, including which instruction sets and optimization it may generate. "code" is one of the following:

  - CORE-AVX2

  - AVX

  - SSE4.2

  - SSE2

- `-xHost` : Applies the highest level of vectorization supported depending on the processor where the compilation is performed. The login nodes may not have the same level of support as the compute nodes. So this option is to be used only if the compilation is done on the targeted compute nodes.

- `-axcode` : Tells the compiler to generate a single executable with multiple levels of vectorization. "code" is a comma-separated list of instructions sets.

The default level of vectorization is sse2. However, it is only be activated for optimization level `-O2` and more.

- `-vec-report[=n]` : depending on the value of n, the option `-vec-report` enables information reports by the vectorizer.

> **Warning:** A code compiled for a given instruction set will not run on a processor that only supports a lower instruction set

Default compilation flags

By default each of the Intel compiler provide the `-sox` option which allows to save all the options provided at the compilation time in the comment section of the ELF binary file. To display the comment section :

```
$ icc -g -O3 hello.c -o helloworld
$ readelf -p .comment ./helloworld
 String dump of section '.comment':
 [     0]  GCC: (GNU) <x.y.z> (Red Hat <x.y.z>)
 [    2c]  -?comment:Intel(R) C Intel(R) 64 Compiler for applications running on␣
↪Intel(R) 64, Version <x.y.z> Build <XXXXXX>  : hello.c : -sox -g -O3 -o helloworld
```

14.2.2 GNU

Compiler flags

Basic flags:

- `-o exe_file`: names the executable `exe_file`
- `-c`: generates the corresponding object file, without creating an executable.
- `-g`: compiles with the debug symbols.
- `-I dir_name`: specifies the path where the include files are located.
- `-L dir_name`: specifies the path where the libraries are located.
- `-l bib`: asks to link the `libbib.a` library

To tell the compiler to conform to a specific language standard (g++/gcc/gfortran) :

- `-std=val`: where **val** can take the following values (cf. **man gcc/g++/gfortran**)
    - c++14 : Enables support for the 2014 ISO C++ standard features.
    - C99 : Conforms to The ISO/IEC 9899:1999 International Standard.
    - f03: Tells the compiler to issue messages for language elements that are not standard in Fortran 2003
    - f08: Tells the compiler to issue messages for language elements that are not standard in Fortran 2008
    - …

Below are some specific flags for the **gfortran** commands.

Debugging:

- `-Wall`: short for "warn about all", warns about usual causes of bugs, such as having a subroutine or function named like a built-in one, or passing the same variable as an intent(in) and an intent(out) argument of the same subroutine
- `-Wextra`: used with `-Wall`, warns about even more potential problems, like unused subroutine arguments
- `-w`: inhibits all warning messages (not recommended)
- `-Werror`: considers any warning as an error

Optimization flags

Compilers provide many optimization options: this section describes them.

Basic optimization options :

- `-O0`, `-O1`, `-O2`, `-O3`: optimization levels - default: `-O0`

Some options allow usage of specific set of instructions for Intel processors, to optimize code behavior. They are compatible with most Intel processors. The compiler will try to use them if the processor allows them.

- `-mavx2` / `-mno-avx2` : Switch on or off the usage of said instruction set.

- `-mavx` / `-mno-avx` : idem.

- `-msse4.2` / `-mno-sse4.2` : idem.

14.3 Available numerical libraries

14.3.1 MKL library

The Intel MKL library is integrated in the Intel package and contains:

- BLAS, SparseBLAS;

- LAPACK, ScaLAPACK;

- Sparse Solver, CBLAS ;

- Discrete Fourier and Fast Fourier transform

If you don't need ScaLAPACK:

```
$ module load mkl
$ ifort -o myexe myobject.o ${MKL_LDFLAGS}
```

If you need ScaLAPACK:

```
$ module load scalapack
$ mpif90 -o myexe myobject.o ${SCALAPACK_LDFLAGS}
```

We provide multi-threaded versions for compiling with MKL:

```
$ module load feature/mkl/multi-threaded
$ module load mkl
$ ifort -o myexe myobject.o ${$MKL_LDFLAGS}
```

```
$ module load feature/mkl/multi-threaded
$ module load scalapack
$ mpif90 -o myexe myobject.o ${SCALAPACK_LDFLAGS}
```

To use multi-threaded MKL, you have to set the OpenMP environment variable `OMP_NUM_THREADS`.

We strongly recommend you to use the `MKL_XXX` and `SCALAPACK_XXX` environment variables made available by the mkl and scalapack modules.

14.3.2 FFTW

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, with an arbitrary input size, and with both real and complex data. It is provided by the **fftw3/gnu** module. The variables FFTW3_CFLAGS and FFTW3_LDFLAGS should be used to compile a code using fftw routines.

```
$ module load fftw3/gnu
$ icc ${FFTW3_CFLAGS} -o test example_fftw3.c ${FFTW3_LDFLAGS}
$ ifort ${FFTW3_FFLAGS} -o test example_fftw3.f90 ${FFTW3_LDFLAGS}
```

Intel MKL also provides Fourier transform functions. FFTW3 wrappers are able to link programes so that they can use Intel MKL Fourier transforms instead of the FFTW3 library, without changing the source code. The correct compiling options are provided by the **fftw3/mkl** module.

```
$ module load fftw3/mkl
$ icc ${FFTW3_CFLAGS} -o test example_fftw3.c ${FFTW3_LDFLAGS}
$ ifort ${FFTW3_FFLAGS} -o test example_fftw3.f90 ${FFTW3_LDFLAGS}
```

14.4 Compiling for Skylake

With the `-ax` option, **icc** and **ifort** can generate code for several architectures.

For example, from a Broadwell login nodes, you can generate an executable with both AVX2 (Broadwell) and AVX512 (Skylake) instructions set. To do so, you need to add the `-axCORE-AVX2,CORE-AVX512` option to **icc** or **ifort**.

An executable compiled with `-axCORE-AVX2,CORE-AVX512` can be run on both Broadwell and Skylake as the best instruction set available on the architecture will be chosen.

14.5 Compiling for Rome/Milan

With the `-m` option, **icc** and **ifort** can generate specific instruction sets for Intel and non-Intel processors.

AMD Rome and Milan processors are able to run AVX2 instructions. To generate an AVX2 instructions for AMD processors, you need to add the `-mavx2` option to **icc** or **ifort**. An executable compiled with `-mavx2` can run on both AMD and Intel processors.

Note:

- The `-mavx2` option is compatible with **gcc**.
- Both `-mavx2` and `-axCORE-AVX2,CORE-AVX512` options can be used simultaneously with **icc** and **ifort** to generate both specific instructions for Intel processors and more generic instructions for AMD processors.

PARALLEL PROGRAMMING

15.1 MPI

The MPI (Message Passing Interface) standard is an API for processes that need to send, wait or receive messages. A full documentation of all the implemented functions can be found in the MPI Standard documentation.

15.1.1 MPI implementations

The supercomputer comes with a default MPI implementation provided by the manufacturer, which is Open MPI supported as well as WI4MPI wrapper. These MPI stacks are strongly recommended and fully supervised by an expert team on the computing center. We ensure then end-to-end support including installation, configuration, performance optimization and issue resolution for e.g. scalability or debugging.

For other MPI libraries like MPICH or Intel MPI, only "best effort" support is provided to access them "as is" for ensuring availability, but without full guarantees on performance. It relies mainly on existing knowledge, documentation, and community support.

Other existing implementations include MVAPICH2 or Platform MPI but all are not made available on the cluster. To see a list of implementations available, use the command **module avail mpi**.

The default MPI implementation is loaded in your environment at connexion time.

15.1.2 Compiling MPI programs

You can compile and link MPI programs using the wrappers **mpicc**, **mpic++**, **mpif77** and **mpif90**. Those wrappers actually call basic compilers but add the correct paths to MPI include files and linking options to MPI libraries.

For example to compile a simple program using MPI:

```
$ mpicc -o mpi_prog.exe mpi_prog.c
$ mpic++ -o mpi_prog.exe mpi_prog.cpp
$ mpif90 -o mpi_prog.exe mpi_prog.f90
```

To see the full compiling call made by the wrapper, use the command **mpicc -show**.

15.1.3 Wi4MPI

Interoperability between MPI implementations is usually not possible because each one has a specific Application Binary Interface (ABI). To overcome this, we have Wi4MPI. Wi4MPI provides two modes (Interface and Preload) with the same promise, one compilation for several run on different MPI implementations (OpenMPI, IntelMPI).

Interface

In this mode, Wi4MPI works as an MPI implementation.

```
$ module load mpi/wi4mpi
feature/wi4mpi/to/openmpi/x.y.z (WI4MPI feature to openmpi x.y.z)
wi4mpi/a.b.c (Wi4MPI with openmpi/x.y.z)
$ mpicc hello.c -o hello
$ ccc_mprun ./hello
```

By default, Wi4MPI is set to run application under OpenMPI. To choose the runtime MPI implementation please proceed as follow:

```
$ module switch feature/wi4mpi/to/openmpi/x.y.z feature/wi4mpi/to/intelmpi/a.b.c
```
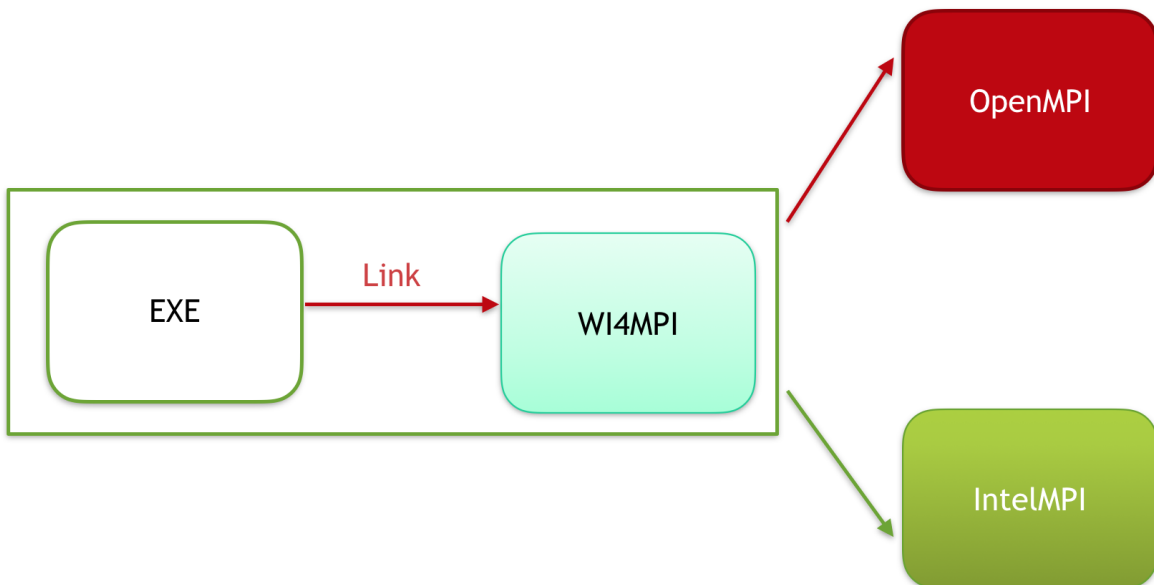

Fig. 1: interface

Preload

In this mode, Wi4MPI works as a plugin.

```
$ module load mpi/wi4mpi
feature/wi4mpi/to/openmpi/x.y.z (WI4MPI feature to openmpi x.y.z)
wi4mpi/a.b.c (WI4MPI with openmpi/x.y.z)
```

This time the MPI implementation used for the compilation of the application needs to be provided as follow:

```
$ module load feature/wi4mpi/from/openmpi/x.y.z
$ module list
1) ccc                4) licsrv/intel          7) fortran/intel/x.y.z  10) mkl/x.y.z      ␣
→   13) hwloc/x.y.z                  16) mpi/wi4mpi/x.y.z
2) datadir/own        5) c/intel/x.y.z    8) feature/mkl/lp64          11) intel/x.y.z    ␣
→   14) flavor/openmpi/cea          17) feature/wi4mpi/from/openmpi/x.y.z
3) dfldatadir/own    6) c++/intel/x.y.z  9) feature/mkl/sequential    12) flavor/wi4mpi/
→standard  15) feature/wi4mpi/to/openmpi/x.y.z
$ ccc_mprun exe_OpenMPI
```
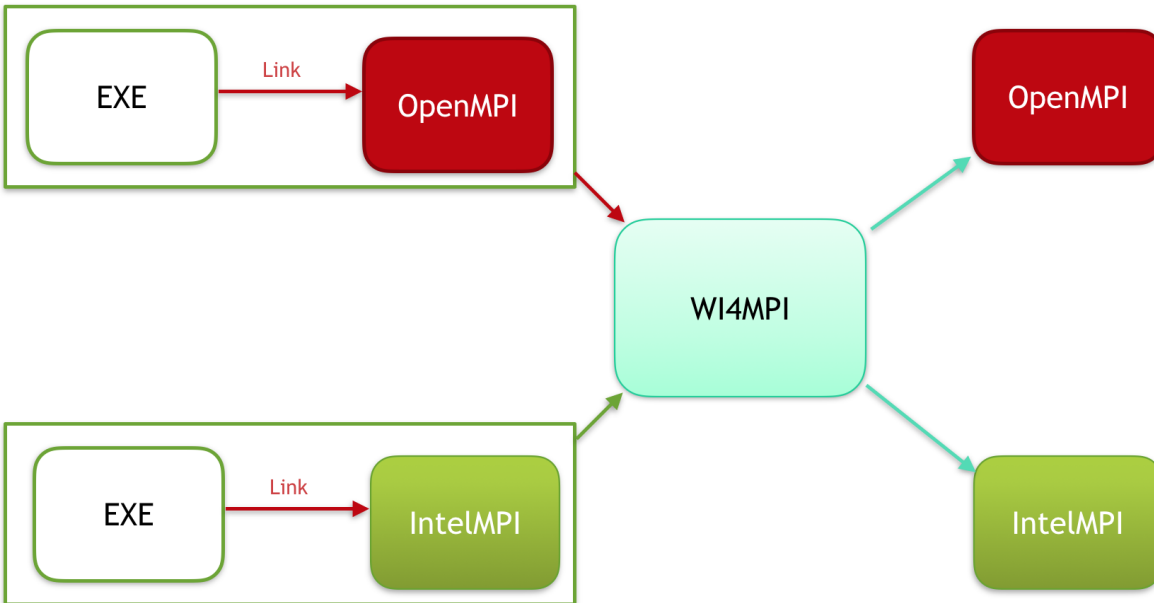

To see all available features:

```
$ module avail feature/wi4mpi
```

To know more about Wi4MPI please visit the cea-hpc github.

15.1.4 Tuning MPI

OpenMPI

MCA Parameters

OpenMPI can be tuned with parameters. The command **ompi_info -a** gives you a list of all parameters and their description.

```
$ ompi_info -a
(...)
  MCA mpi: parameter "mpi_show_mca_params" (current value: <none>, data source: default␣
→value)
           Whether to show all MCA parameter values during MPI_INIT or not (good for
           reproducibility of MPI jobs for debug purposes). Accepted values are all,
           default, file, api, and environment - or a comma delimited combination of them
(...)
```

Theses parameters can be modified with environment variables set before the **ccc_mprun** command. The form of the corresponding environment variable is OMPI_MCA_xxxxx where *xxxxx* is the parameter.

```
#!/bin/bash
#MSUB -r MyJob_Para               # Job name
#MSUB -n 32                       # Number of tasks to use
#MSUB -T 1800                     # Elapsed time limit in seconds
#MSUB -o example_%I.o             # Standard output. %I is the job id
#MSUB -e example_%I.e             # Error output. %I is the job id
#MSUB -q partition                # Partition name
#MSUB -A <project>                # Project ID

set -x
cd ${BRIDGE_MSUB_PWD}
export OMPI_MCA_mpi_show_mca_params=all
ccc_mprun ./a.out
```

For more information on MCA parameters, check out the *tuning* part of the openmpi FAQ.

Predefined MPI profiles

Some common MCA parameters are defined in several features (openmpi) to simplify their usage.

Here are those modules:

```
$ module avail feature/openmpi
---------------- /opt/Modules/default/modulefiles/applications ----------------
---------------- /opt/Modules/default/modulefiles/environment ----------------
feature/openmpi/big_collective_io  feature/openmpi/gnu  feature/openmpi/mxm  feature/
→openmpi/performance  feature/openmpi/performance_test
```

Here is their description and their limitation(s):

- **performance**

Description : Improve communication performances in classic applications (namely those with fixed communication scheme). Limitation : Moderately increase the memory consumption of the MPI layer.

- **big_collective_io**

    Description : Increase data bandwidth when accessing big files on lustre file system. Limitation : Only useful when manipulating big files through MPI_IO and derivated (parallel hdf5, etc.).

- **collective_performance**

    Description : Improve the performance of several MPI collective routines by using the GHC feature developed by BULL. Limitation : Improvements may not be seen on small cases. Namely change the order in which MPI reduction operations are performed and may impact the reproductability and/or the numerical stability of very sensible systems.

- **low_memory_footprint**

    Description : Reduce the memory consumption of the MPI layer. Limitation : May have strong impact over communication performances. Should only be used when you are near the memory limits.

Compiler Features

By default, MPI wrappers use Intel compilers. To use GNU compilers, you need to set the following environment variables:

- `OMPI_CC` for C
- `OMPI_CXX` for C++
- `OMPI_F77` for fortran77
- `OMPI_FC` for fortran90

For example:

```
$ export OMPI_CC=gcc
$ mpicc -o test.exe test.c
```

It's also possible to use the feature **feature/openmpi/gnu** which set all variables define above at GNU compilers (gcc, g++, gfortran).

```
$ module load feature/openmpi/gnu
load module feature/openmpi/gnu (OpenMPI profile feature)
```

IntelMPI

Compiler Features

By default, MPI wrappers use Intel compilers. To use GNU compilers, you need to set the following environment variables:

- `I_MPI_CC` for C
- `I_MPI_CXX` for C++
- `I_MPI_F77` for fortran77
- `I_MPI_FC` for fortran90

For example:

```
$ export I_MPI_CC=gcc
$ mpicc -o test.exe test.c
```

It's also possible to use the feature **feature/intelmpi/gnu** which set all variables define above at GNU compilers (gcc, g++, gfortran).

```
$ module load feature/intelmpi/gnu
load module feature/intelmpi/gnu (IntelMPI profile feature)
```

15.2 OpenMP

OpenMP is an API that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior. More information and a full documentation can be found on the official website.

15.2.1 Compiling OpenMP programs

The Intel and GNU compilers both support OpenMP. Use the `-fopenmp` flag to generate multi-threaded code with those compilers.

Note: `-openmp` works only for intel whereas `-fopenmp` works with both Intel and GNU compilers.

```
$ icc -fopenmp -o openmp_prog.exe openmp_prog.c
$ icpc -fopenmp -o openmp_prog.exe openmp_prog.cpp
$ ifort -fopenmp -o openmp_prog.exe openmp_prog.f90
```

15.2.2 Intel thread affinity

By default, threads inherit the same affinity than their parent process (see *Process distribution, affinity and binding* for more information). For example, if the parent process has been allocated 4 cores, the OpenMP threads spawned by this process will be allowed to run freely on those 4 cores.

To set a more specific binding for threads among the allocated cores, Intel provides environment variables: `KMP_AFFINITY`

The values given to `KMP_AFFINITY` should be a comma-separated list of the following keywords:

- `verbose`: prints messages concerning the supported affinity.

- `granularity=`: specifies whether to pin OpenMP threads to physical cores (`granularity=core`, this is the default) or pin to logical cores (`granularity=fine`). This is only effective on nodes that support SMT (Simultaneous Multithreading, as hyperthreading on Intel architecture for instance).

- `compact`: assigns the threads as close as possible together.

- `scatter`: distributes the threads as evenly as possible across the entire system.

15.3 GPU-accelerated computing

This section is about GPU-accelerated compute jobs. For information about GPU-accelerated remote desktop visualization service, please refer to the *Interactive access* section.

15.3.1 CUDA

CUDA may refer to

- The **CUDA Language Extentions**: A set of language extentions for C, C++ and Fortran.
- The **CUDA Driver**: A low level API for interacting with GPU devices.
- The **CUDA Runtime API**: A higher-level programming interface for developping GPU-accelerated code.
- The **CUDA Runtime library**: The implementation of the CUDA Runtime API
- The **CUDA Toolkit**: A set of tools and libraries distributed by NVIDIA including the CUDA runtime Libraries.

NVIDIA graphics cards are operated through the CUDA driver. It is a low level API offering fine-grained control which requires explicit management of cuda code, CUDA module loading, and execution context. The CUDA runtime API is a higher level, more easy-to-use programming interface. Most applications do not use the driver API as they do not need the finer level of control. The CUDA runtime API provides all functions required for running GPU computations on NVIDIA graphics cards such as device management, thread management, memory allocation and copying, event management and execution control. Visit the online CUDA documentation for more informations.

The CUDA programming model includes a number of language extentions to C and C++. CUDA Fortran directive-based compiler supports similar functionnalities. OpenMP and OpenACC are other programming models that can be used for writing efficient GPU-accelerated programs and are supported by the NVIDIA and GCC compilers.

15.3.2 Overview of NVHPC

The NVHPC toolkit includes a number of compilers and libraries designed for GPU programming. It is available though the nvhpc module :

```
$ module load nvhpc
```

NVHPC includes the **nvc**, **nvfortran**, **nvc++** and **nvcc** compilers.

nvc

> **nvc** is a C11 compiler for NVIDIA GPUs and AMD, Intel, OpenPOWER, and Arm CPUs. It invokes the C compiler, assembler, and linker for the target processors with options derived from its command line arguments. **nvc** supports ISO C11, supports GPU programming with OpenACC, and supports multicore CPU programming with OpenACC and OpenMP.

nvc++

> **nvc++** is a C++17 compiler for NVIDIA GPUs and AMD, Intel, OpenPOWER, and Arm CPUs. It invokes the C++ compiler, assembler, and linker for the target processors with options derived from its command line arguments. **nvc++** supports ISO C++17, supports GPU and multicore CPU programming with C++17 parallel algorithms, OpenACC, and OpenMP.

nvfortran

> **nvfortran** is a Fortran compiler for NVIDIA GPUs and AMD, Intel, OpenPOWER, and Arm CPUs. It invokes the Fortran compiler, assembler, and linker for the target processors with options derived from its command line arguments. **nvfortran** supports ISO Fortran 2003 and many features of ISO Fortran 2008,

supports GPU programming with CUDA Fortran, and GPU and multicore CPU programming with ISO Fortran parallel language features, OpenACC, and OpenMP.

nvcc

nvcc is a CUDA C/C++ compiler driver for NVIDIA GPUs. **nvcc** accepts a range of conventional compiler options, such as for defining macros and include/library paths, and for steering the compilation process. All non-CUDA compilation steps are forwarded to a C++ host compiler.

NVHPC also comes with a set of mathematic libraries such as **cublas**, **cufft**, and **nccl**.

CUDA Math API

The CUDA math API supports the usual intrinsics and mathematical functions for integer, 64-bit, 32-bit and 16-bit floats.

cuBLAS

The **cuBLAS** library is an implementation of BLAS (Basic Linear Algebra Subprograms) on top of the NVIDIA CUDA runtime. It allows the user to access the computational resources of NVIDIA Graphical Processing Unit (GPU), but does not auto-parallelize across multiple GPUs.

cuSPARSE

cuSPARSE implements linear algebra for sparse vectors and matrices. It supports **coo**, **csr** and **csc** sparse formats.

cuSOLVER

Based on the **cuBLAS** and **cuSPARSE** libraries, **cuSOLVER** implements two high-level API for dense and sparse linear algebra. The **cuSolver API** provides single-GPU LAPACK-like features. The **cuSolverMG API** provides an implementation of the ScaLAPACK API for single node multi-GPU execution.

NVBLAS

NVBLAS is another implementation of most level 3 BLAS routines. It supports dynamic execution on multiple GPUs. It is built on top of the cuBLAS library.

cuFFT

The **CUDA Fast Fourier Transform Library** supports batched, multi-GPU, 1D, 2D and 3D transforms. Il implements and extends the FFTW3 API.

NCCL

The **NVIDIA Collective Communications Library** (**NCCL**, pronounced "Nickel") is a library providing inter-GPU communication primitives that are topology-aware and can be easily integrated into applications.

NCCL implements both collective communication and point-to-point send/receive primitives. It is not a full-blown parallel programming framework; rather, it is a library focused on accelerating inter-GPU communication.

The documentation for all of these can be found at nvidia.com: NVIDIA HPC Documentation

15.3.3 Using NVHPC libraries

nvc, **nvcc** and **nvfortran** compilers accept special compilation flags in order to enable compilation with these libraries using the following syntax, which will use the libaries available from *-L* command line arguments or from `LD_LIBRARY_PATH`

```
-cudalib[=cublas|cufft:{callback}
→|cufftw|curand|cusolver|cusparse|cutensor|nvblas|nccl|nvshmem|nvlamath]
```

In order to compile and link your code using the libraries included in nvhpc, you need to load nvhpc and/or its dependencies so that the required libraries are available from `LD_LIBRARY_PATH`

```
$ module load nvhpc

$ module show math/nvidia
[...]
append-path LD_LIBRARY_PATH     /ccc/products/nvhpc-21.2/system/default/Linux_x86_64/21.
→2/math_libs/lib64
[...]
setenv      MATH_NVIDIA_LIBDIR /ccc/products/nvhpc-21.2/system/default/Linux_x86_64/21.
→2/math_libs/lib64
[...]


$ ls $MATH_NVIDIA_LIBDIR
libcublas.so    libcublasLt.so  libcublasLt.so.11
libcublasLt.so.11.4.2.10064     libcublasLt_static.a
libcublas.so.11 libcublas.so.11.4.2.10064
libcublas_static.a              libcufft.so
libcufftw.so    libcufftw.so.10 libcufftw.so.10.4.2.58
libcufftw_static.a              libcufft.so.10
libcufft.so.10.4.2.58           libcufft_static.a
libcufft_static_nocallback.a    libcurand.so
libcurand.so.10 libcurand.so.10.2.4.58  libcurand_static.a
[...]
```

If you want to use some libraries included in NVHPC that are not direcly available using such environment variables, use paths relative to variables such as `NVHPC_ROOT` or `MATH_NVIDIA_ROOT`

```
$ NVSHMEM_LIBDIR=$MATH_NVIDIA_ROOT/../comm_libs/nvshmem/lib/
$ LD_LIBRARY_PATH=$LD_LIBARY_PATH:$NVSHMEM_LIBDIR
```

You may also use **-I** or **-L** compilation and linking options to specify the paths of includes and libraries directories to your favorite C/C++ compiler.

15.3.4 Using NVCC

NVCC is a helper driver for compiling CUDA C/C++ code. It handles a number source code transformation, invokes the underlying CUDA compiler as well as a general purpose C++ host compiler (**g++** by default) for compiling non-CUDA code and preprocessing. Its purpose is to use the development of CUDA applications.

```
$ module load nvhpc
$ nvcc -o test_cuda test.cu
```

GPU Hardware Architectures, and the set of features supported by a particular iteration over a general architecture are identified by the *compute capabilities* of a target GPU. As such, compute capabilities 7 refer to the Volta architecture while compute capabilities 8 refer to the Amper architecture.

Use the following command to list the GPU architectures as well as the virtual device architectures supported by the NVCC compiler.

```
$ nvcc --list-gpu-code --list-gpu-arch
```

Use the `-gencode` argument to NVCC to specify the target GPU architecture as well as the virtual device architecture. You may specify several levels of support if you need the code to run on different GPU generations. For generating GPU code for both V100 and A100 GPUs, use

```
$ nvcc -gencode arch=compute_72,code=sm_72 -gencode arch=compute_80,code=sm_80 -o test_
↪cuda test.cu
```

The **nvcc** command uses C and C++ compilers underneath. Compiling with **nvcc -v** will show the detail of the underlying compiling calls. By default, the GNU compilers are used. To change the underlying compiler, use the `-ccbin` option:

```
$ module load nvhpc
$ module load intel
$ nvcc -v -ccbin=icc -o test_cuda test.cu
```

Most GPU codes are partially composed of basic sequential codes in separate files. They may be compiled separately:

```
$ ls
   cuda_file.cu    file.c
$ icc -o file.o -c file.c
$ nvcc -ccbin=icc -o cuda_file.o -c cuda_file.cu
$ icc -o test_cuda -L${CUDA_LIBDIR} -lcudart file.o cuda_file.o
```

15.3.5 CUDA fortran

The CUDA Fortran language extentions allow programming for GPU in Fortran using similar idioms to C/C++ CUDA. To compile CUDA Fortran code, use the **nvfortran** compiler as described in section *Compiling OpenACC and OpenMP code for GPU* with the `-cuda` option in place of the `-acc` option.

15.3.6 Compiling OpenACC and OpenMP code for GPU

Based on the LLVM compiler, **nvc**, **nvc++** and **nvfortran** can be used for compiling C, C++ and Fortran code enriched with OpenMP or OpenACC directives gor execution on NVIDIA GPUs.

They determine the source input type by examining the filename extentions, and use the following conventions:

fortran sources

    filename.f

        indicates a Fortran source file.

    filename.F

        indicates a Fortran source file that can contain macros and preprocessor directives (to be preprocessed).

    filename.FOR

        indicates a Fortran source file that can contain macros and preprocessor directives (to be preprocessed).

filename.F90

indicates a Fortran 90/95 source file that can contain macros and preprocessor directives (to be preprocessed).

filename.F95

indicates a Fortran 90/95 source file that can contain macros and preprocessor directives (to be preprocessed).

filename.f90

indicates a Fortran 90/95 source file that is in freeform format.

filename.f95

indicates a Fortran 90/95 source file that is in freeform format.

filename.cuf

indicates a Fortran 90/95 source file in free format with CUDA Fortran extensions.

filename.CUF

indicates a Fortran 90/95 source file in free format with CUDA Fortran extensions and that can contain macros and preprocessor directives (to be preprocessed).

C/C++ sources

filename.c

indicates a C source file that can contain macros and preprocessor directives (to be preprocessed).

filename.C

indicates a C++ source file that can contain macros and preprocessor directives (to be preprocessed).

filename.i

indicates a preprocessed C or C++ source file.

filename.cc

indicates a C++ source file that can contain macros and preprocessor directives (to be preprocessed).

filename.cpp

indicates a C++ source file that can contain macros and preprocessor directives (to be preprocessed).

See NVIDIA.com - HPC Compilers User's Guide.

Here is an example of compilation using the **nvfortran** compiler, highlighting a few useful compilation options:

```
$ nvfortran -acc=gpu,host -gpu=cc70,cc80,time -Minfo=accel -fast -c src/example_file.f90␣
↪-o bin/example_file.o -module bin
$ nvfortran -acc=gpu,host -gpu=cc70,cc80,time -Minfo=accel -fast -o bin/example_bin bin/
↪example_file.o ccc_mprun ./example_bin
```

-acc=gpu,host

enables OpenACC directives for execution on GPU when at least one such device is available, or sequantial CPU execution otherwise.

-mp=gpu

enables OpenMP directives for execution on GPU devices.

-mp=multicore

enables OpenMP directives for execution on CPU.

-gpu=cc70,cc80

specifies that both V100 and A100 GPU architectures should be targeted for GPU executable code.

-gpu=time

enables light profiling of the accelerator regions and generated kernels. A profiling report is printed at

the end of the resulting program execution. This can also be achieved by setting the environment variable `NVCOMPILER_ACC_TIME` to a non-zero value during compilation.

-Minfo=accel

enables output of information about accelerator compute kernel compilation. Use `-Minfo=all` for an even more verbose compilation output, including general and loop-specific optimization information.

-fast

enables the most impactful optimization options. Use `-O2` or lower optimization level instead for development phases since `-fast` disables stack flame generation and makes execution analysis and debugging generally more complex.

Other useful preprocessing, compiling and linking options such as `-Mfree` (assume free-format fortran source), are available. Please refer to `man <compiler>`, `<compiler> --help`, or NVIDIA's HPC Compilers User's Guide for more information

15.3.7 Compiling GPU-accelerated MPI programs

The `mpi/openmpi` module is available with NVHPC compilers support. Loading the following sequence of modules enables using the **mpicc**, **mpic++** and **mpif90** compiler wrappers with the NVHPC compilers:

```
$ module load nvhpc
$ module load mpi/openmpi
```

You may then use **mpif90** in place of **nvfortran** for compiling and linking distributed, GPU-accelerated mpi applications.

Accelerated MPI routines are available for transfering data between GPUs inside and across nodes. Specific configurations are loaded when loading MPI with nvhpc on GPU partitions to enable *GPU Direct RDMA*, *Device to Device*, and *CUDA-aware MPI*, enabling specific programming practices such as initiating MPI communications using *device pointers* in cuda, or use of the `host data use device` OpenACC directive.

15.3.8 Running GPU applications

This section is about GPU-accelerated compute jobs. For information about GPU-accelerated remote desktop visualization service, please refer to the *Interactive access* section. Programs compiled with cuda are started normally and will use the cuda runtime libraries and drivers in order to offload computations to the GPUs. GPU MPI programs are started normally using **ccc_mprun**.

- **Simple one GPU job**

```bash
#!/bin/bash
#MSUB -r GPU_Job            # Job name
#MSUB -n 1                  # Total number of tasks to use
#MSUB -c 32                 # Assuming compute nodes have 128 cores and 4 GPUs
#MSUB -T 1800               # Elapsed time limit in seconds
#MSUB -o example_%I.o       # Standard output. %I is the job id
#MSUB -e example_%I.e       # Error output. %I is the job id
#MSUB -q <partition>        # Partition name
#MSUB -A <project>          # Project ID
set -x
cd ${BRIDGE_MSUB_PWD}
module load nvhpc
ccc_mprun ./a.out
```

Assuming 4 GPUs per node, each GPU reservation allocates 1/4th of a node. When all 4 GPUs are reserved, a full node is allocated. For job allocations using more than one full node, additionnal nodes are allocated fully. (see *Process distribution, affinity and binding*).

Use the `#MSUB -c` option to tune the number of processes per allocated nodes for reducing the number of processes per CPU core.

Depending on the application, if a node has 128 cpus and 4 GPUs, use:

- use `-c 128` to run a single process per node (4 GPU per process)

- use `-c 32` to run 4 processes per node (1 GPU per process)

- use `-c 1` to run a process per CPU core (32 processes per GPU)

Note that `#MSUB -c` option let choose the number of GPUs needed for the job:

- `-n 1 -c 1` option doesn't allocate GPU

- `-n 1 -c 32` option allocates 1 GPU

- `-n 1 -c 33` option allocates 1 GPU

- `-n 1 -c 64` option allocates 2 GPUs

- `-n 1 -c 96` option allocates 3 GPUs

- `-n 1 -c 128` option allocates 4 GPUs

The environment variable `CUDA_VISIBLE_DEVICE` controls the number and id of the GPUs visible from the current process. For example, in order to use 4 processes per node on 2 nodes, each process managing a single GPU, you may use the following job script.

```bash
#!/bin/bash
#MSUB -r MPI_GPU_Job      # Job name
#MSUB -n 8                # Total number of tasks to use
#MSUB -c 32               # space out processes
#MSUB -T 1800             # Elapsed time limit in seconds
#MSUB -o example_%I.o     # Standard output. %I is the job id
#MSUB -e example_%I.e     # Error output. %I is the job id
#MSUB -q <partition>      # Partition name
#MSUB -A <project>        # Project ID
set -x
set -x
cd ${BRIDGE_MSUB_PWD}
module load nvhpc mpi
ccc_mprun ./set_visible_device.sh ./myprogram
```

With the set_visible_device.sh script setting the `CUDA_VISIBLE_DEVICE` environment variable to a different value for each mpi process. It may use available information from each of the MPI rank being started to set **CUDA_VISIBLE_DEVICE** for that particular process.

This example implementation associates each GPU with the processes according to their position in Slurm's node-local rank numbering. This may produce sub-optimal memory exchange in cases where the GPU enumeration order (as given by the `SLURM_STEP_GPUS` environment variable) does not follow the CPU enumeration order (assuming MPI process enumeration order does follow CPU cores enumeration order). Refer to the output of the **nvidia-smi topo -m** command to check the actual CPU and NUMA affinity of each nvidia graphics device.

```bash
#!/bin/bash
# set_visible_device.sh
```

(continues on next page)

```
NGPUS=$(echo ${SLURM_STEP_GPUS//[^,]/} | wc -c)
# CUDA_VISIBLE_DEVICES = round_down( local_process_id_in_node / processes_per_gpu )
export CUDA_VISIBLE_DEVICES=$(( $SLURM_LOCALID * $SLURM_CPUS_PER_TASK * $NGPUS / ${SLURM_
→JOB_CPUS_PER_NODE%(*)} ))
exec $*
```

For more general information about process placement and binding, please refer to the *Process distribution, affinity and binding* section

15.3.9 Debugging, profiling and performance analysis

nvidia-smi

> **nvidia-smi** is a useful monitoring and management tool. It shows the available GPUs, running processes, memory usage. Use `nvidia-smi topo -m` to display GPUDirect interconnection topology.
>
> Note that repeatedly polling GPU usage through repeated calls to nvidia-smi or usage of the `--loop` or `--loop-ms` options may impact performance.
>
> The `dmon` and `daemon` options may help setting up monitoring of GPUs usage.

CUDA-GDB

> **CUDA-GDB** is an extention to gdb for debugging C/C++ and Fortran CUDA applications running on actual hardware. It supports program stepping, breakpoints, thread and memory inspection for CUDA code in addition to the usual gdb capabilities regarding CPU code.

CUPTI

> **CUPTI** is the CUDA Profiling Tools Interface. It enables the creation of profiling and tracing tools that target CUDA applications. Its documentation can be found online.

Nsight Systems

> Please refer to the *the dedicated section*.

Nsight Compute

> Please refer to the *the dedicated section*.

RUNTIME TUNING

16.1 Memory allocation tuning

16.1.1 Feature TBB malloc

Intel TBB (Threading Building Blocks) presents an API (Application Program Interface) that take full advantage of multicore performance. Among this API, there is some allocator that could be found in the GNU C Library as malloc or calloc but performing in a more efficient way inside TBB. The easiest way for an application to benefit from it (without any changes within it) is to provide the following librairies to the environment variable LD_PRELOAD, or to use the **feature** available on Irene as follow.

```
$ module load feature/optimization/tbb_malloc
```

Now, at each use of the command **ccc_mprun**, any application will benefit from those allocator.

SEVENTEEN

PROCESS DISTRIBUTION, AFFINITY AND BINDING

17.1 Hardware topology

Hardware topology is the organization of cores, processors, sockets, memory, network cards and accelerators (like graphic cards) in a node. An image of the topology of a node is shown by `lstopo`. This function comes with hwloc, and you can access it with the command `module load hwloc`. For example, a simple graphical representation of a compute node layout can be obtained with:

```
lstopo --no-caches --no-io --of png
```

17.2 Definitions

We define here some vocabulary:

- **Process distribution** : the distribution of MPI processes describes how these processes are spread across the cores, sockets or nodes.

- **Process affinity** : represents the policy of resources management (cores and memory) for processes.

- **Process binding** : a Linux process can be bound (or stuck or pinned) to one or many cores. It means a process and its threads can run only on a given selection of cores.

The default behavior for distribution, affinity and binding is managed by the batch scheduler through the `ccc_mprun` command. SLURM (Simple Linux Utility for Resource Management) is the batch scheduler for jobs on the supercomputer. SLURM manages the distribution, affinity and binding of processes even for sequential jobs. That is why we recommend you to use `ccc_mprun` even for non-MPI codes.

17.3 Process distribution

We present here the most common MPI process distributions. The cluster naturally has a two dimensional distribution. The first level of distribution manages the process layout across nodes and the second level manages process layout within a node, across sockets.

- Distribution across nodes

    - **block** by node: The block distribution method will distribute tasks such that consecutive tasks share a node.

    - **cyclic** by node: The cyclic distribution method will distribute tasks such that consecutive tasks are distributed over consecutive nodes (in a round-robin fashion).

- Distribution across sockets
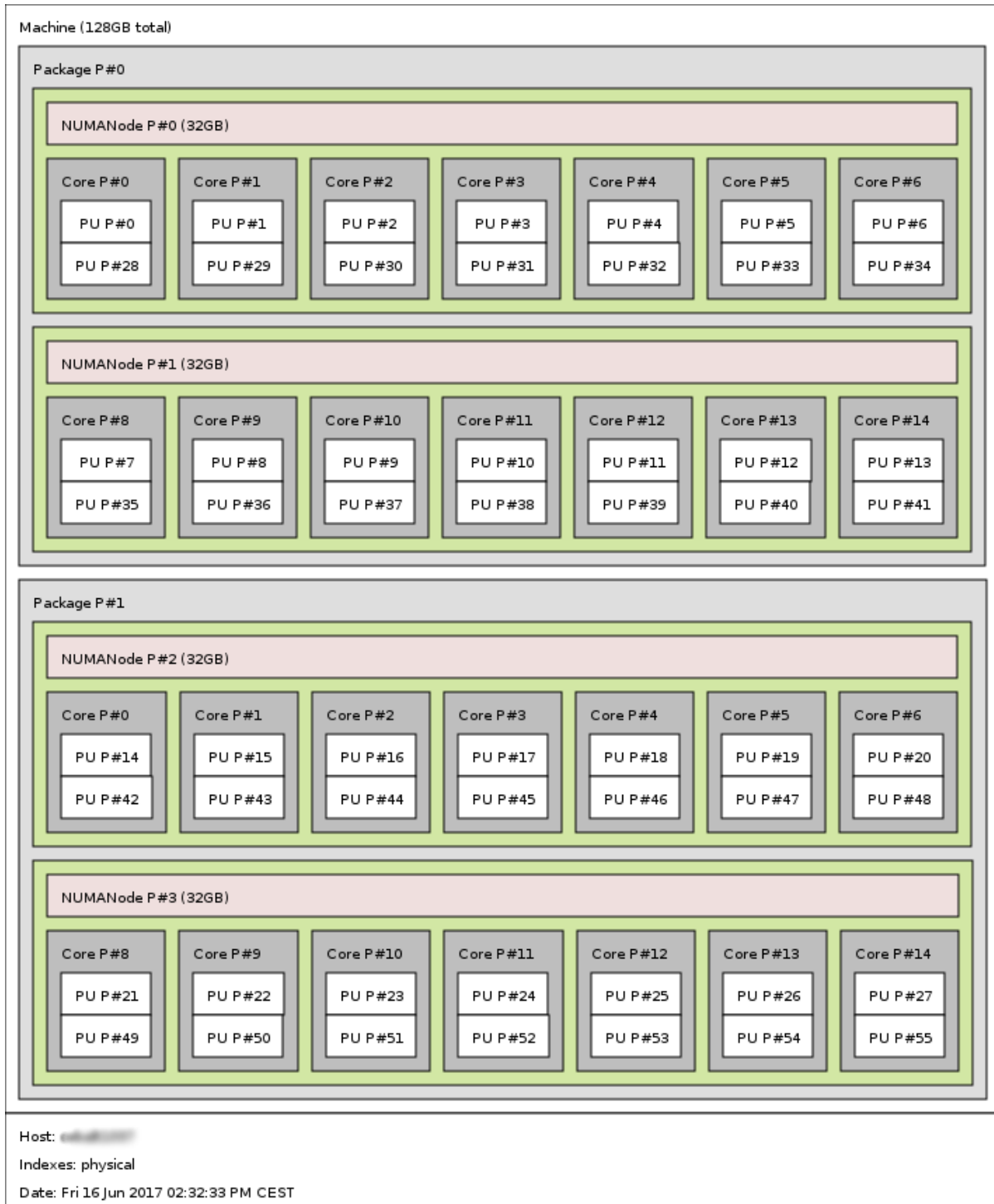
Fig. 1: Hardware topology of a 28 core node with 4 sockets and SMT (Simultaneous Multithreading, as Hyper-threading on Intel architecture)

- **block** by socket: The block distribution method will distribute tasks to a socket such that consecutive tasks share a same socket.
- **cyclic** by socket: The cyclic distribution method will distribute tasks to a socket such that consecutive tasks are distributed over consecutive socket (in a round-robin fashion).

The default distribution on the supercomputer is block by node and block by socket. To change the distribution of processes, you can use the option `-E "-m block|cyclic:[block|cyclic]"` for **ccc_mprun** or for **ccc_msub**. The first argument refers to the node level distribution and the second argument refers to the socket level distribution.

In the following examples, let us consider 2 nodes composed of 2 sockets. Each socket with 8 cores.

- Block by node and block by socket distribution:

```
ccc_mprun -n 32 -E '-m block:block' -A <project> ./a.out
```

- Processes 0 to 7 are gathered on the first socket of the first node
- Processes 8 to 15 are gathered on the second socket of the first node
- Processes 16 to 23 are gathered on the first socket of the second node
- Processes 24 to 31 are gathered on the second socket of the second node

This is the default distribution.

- Cyclic by node and block by socket distribution:

```
ccc_mprun -n 32 -E '-m cyclic:block' -A <project> ./a.out
```

- Process 0 runs on the first socket of the first node
- Process 1 runs on the first socket of the second node
- …

And once the first socket is full, processes are distributed on the next sockets:

- Process 16 runs on the second socket of the first node
- Process 17 runs on the second socket of the second node

17.3. Process distribution 143

– …

```
Node 1
  Socket P#0                          Socket P#1
  Core P#0    Core P#1    Core P#2    Core P#0    Core P#1    Core P#2
     0          2           4           16          18          20
  Core P#3    Core P#4    Core P#5    Core P#3    Core P#4    Core P#5
     6          8          10           22          24          26
  Core P#6    Core P#7                 Core P#6    Core P#7
    12         14                        28          30

Node 2
  Socket P#0                          Socket P#1
  Core P#0    Core P#1    Core P#2    Core P#0    Core P#1    Core P#2
     1          3           5           17          19          21
  Core P#3    Core P#4    Core P#5    Core P#3    Core P#4    Core P#5
     7          9          11           23          25          27
  Core P#6    Core P#7                 Core P#6    Core P#7
    13         15                        29          31
```

- Block by node and cyclic by socket distribution:

```
ccc_mprun -n 32 -E '-m block:cyclic' ./a.out
```

  – Process 0 runs on the first socket of the first node

  – Process 1 runs on the second socket of the first node

  – …

  And once the first node is full, processes are distributed on both sockets of the next node:

  – Process 16 runs on the first socket of the second node

  – Process 17 runs on the second socket of the second node

  – …

Node 1

Socket P#0			Socket P#1		
Core P#0 **0**	Core P#1 **2**	Core P#2 **4**	Core P#0 **1**	Core P#1 **3**	Core P#2 **5**
Core P#3 **6**	Core P#4 **8**	Core P#5 **10**	Core P#3 **7**	Core P#4 **9**	Core P#5 **11**
Core P#6 **12**	Core P#7 **14**		Core P#6 **13**	Core P#7 **15**	

Node 2

Socket P#0			Socket P#1		
Core P#0 **16**	Core P#1 **18**	Core P#2 **20**	Core P#0 **17**	Core P#1 **19**	Core P#2 **21**
Core P#3 **22**	Core P#4 **24**	Core P#5 **26**	Core P#3 **23**	Core P#4 **25**	Core P#5 **27**
Core P#6 **28**	Core P#7 **30**		Core P#6 **29**	Core P#7 **31**	

- Cyclic by node and cyclic by socket distribution:

```
ccc_mprun -n 32 -E '-m cyclic:cyclic' <project> ./a.out
```

  - Process 0 runs on the first socket of the first node

  - Process 1 runs on the first socket of the second node

  - Process 2 runs on the second socket of the first node

  - Process 3 runs on the second socket of the second node

  - Process 4 runs on the first socket of the first node

  - …

Node 1

Socket P#0			Socket P#1		
Core P#0 **0**	Core P#1 **4**	Core P#2 **8**	Core P#0 **2**	Core P#1 **6**	Core P#2 **10**
Core P#3 **12**	Core P#4 **16**	Core P#5 **20**	Core P#3 **14**	Core P#4 **18**	Core P#5 **22**
Core P#6 **24**	Core P#7 **28**		Core P#6 **26**	Core P#7 **30**	

Node 2

Socket P#0			Socket P#1		
Core P#0 **1**	Core P#1 **5**	Core P#2 **9**	Core P#0 **3**	Core P#1 **7**	Core P#2 **11**
Core P#3 **13**	Core P#4 **17**	Core P#5 **21**	Core P#3 **15**	Core P#4 **19**	Core P#5 **23**
Core P#6 **25**	Core P#7 **29**		Core P#6 **27**	Core P#7 **31**	

17.4 Process and thread affinity

17.4.1 Why is affinity important for improving performance ?

Most recent nodes are NUMA (Non-Uniform Memory Access) nodes: they need more time to access some regions of memory than others, because all memory regions are not physically on the same bus.
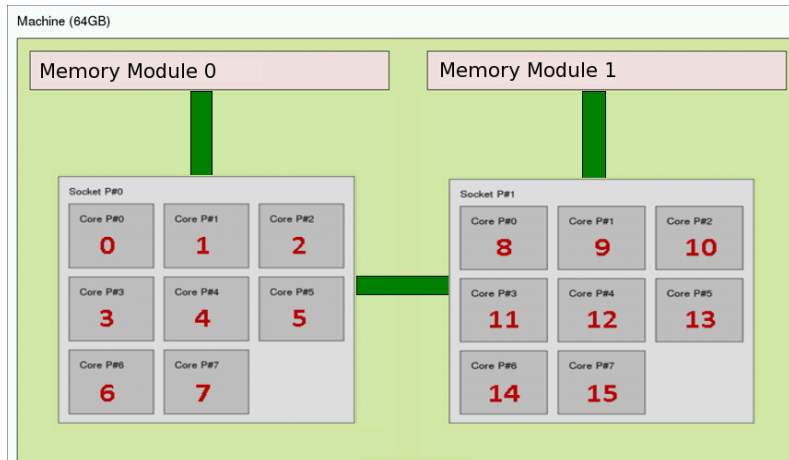
Fig. 2: NUMA node : Memory access

This picture shows that if a data is in the memory module 0, a process running on the second socket like the 9th process will take more time to access the data. We can introduce the notion of *local data* vs *remote data*. In our example, if we consider a process running on the socket 0, a data is *local* if it is on the memory module 0. The data is *remote* if it is on the memory module 1.

We can then deduce the reasons why tuning the process affinity is important:

- Data locality improve performance. If your code uses shared memory (like pthreads or OpenMP), the best choice is to group your threads on the same socket. The shared data should be local to the socket and moreover, the data may stay in the processor's cache.

- System processes can interrupt your process running on a core. If your process is not bound to a core or to a socket, it can be moved to another core or to another socket. In this case, all data for this process have to be moved with the process too and it can take some time.

- MPI communications are faster between processes which are on the same socket. If you know that two processes have many communications, you can bind them to the same socket.

- On Hybrid nodes, the GPUs are connected to buses which are local to a socket. Processes can take more time to access a GPU which is not connected to its socket.

For all these reasons, knowing the NUMA configuration of the partition's nodes is strongly recommended. The next section presents some ways to tune your processes affinity for your jobs.
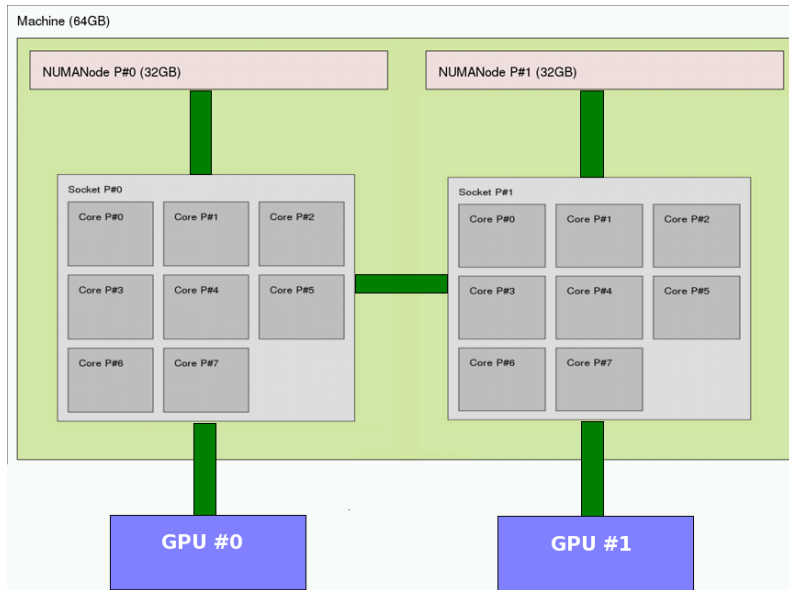
Fig. 3: NUMA node : Example of a hybrid node with GPU

17.4.2 Process/thread affinity

The batch scheduler sets a default binding for processes. Each process is bound to the core it was distributed to, as described in *Process distribution*.

For multi-threaded jobs, the batch scheduler provides the option -c to bind each process to several cores. Each thread created by a process will inherit its affinity. Here is an example of a hybrid OpenMP/MPI code running on 8 MPI processes, each process using 4 OpenMP threads.

```bash
#!/bin/bash
#MSUB -r MyJob_Para              # Job name
#MSUB -n 8                       # Number of tasks to use
#MSUB -c 4                       # Assign 4 cores per process
#MSUB -T 1800                    # Elapsed time limit in seconds
#MSUB -o example_%I.o            # Standard output. %I is the job id
#MSUB -A <project>               # Project ID
#MSUB -q <partition>             # Partition

export OMP_NUM_THREADS=4
ccc_mprun ./a.out
```

The process distribution will take the -c option into account and set the binding accordingly. For example, in the default *block:block* mode:

- Process 0 is bound to cores 0 to 3 of the first node
- Process 1 is bound to cores 4 to 7 of the first node
- Process 2 is bound to cores 8 to 11 of the first node
- Process 3 is bound to cores 12 to 15 of the first node
- Process 4 is bound to cores 0 to 3 of the second node
- …

Note: Since the default distribution is block, with the `-c` option, the batch scheduler will try to gather the cores as close as possible. This usually provides the best performances for multi-threaded jobs. In the previous example, all the cores of a MPI process will be located on the same socket.

Thread affinity may be set even more thoroughly within the process binding. For example, check out the *Intel thread affinity* description.

17.5 Hyper-Threading usage

SMT (Simulatneous Multithreading, as Hyper-threading on Intel architecture) and its implementations are used to improve parallelization of computations. It is activated on compute nodes. Therefore, each physical core appears as two processors to the operating system, allowing concurrent scheduling of two processes or threads per core. Unless specified, the resource manager will only consider physical cores and ignore this feature.

In case an application may positively benefits from this technology, MPI processes or OpenMP threads can be bind to logical cores. Here is the procedure:

- Doubling the processes:

```
#!/bin/bash
#MSUB -q <partition>
#MSUB -n 2
#MSUB -A <projet>
```

<div align="right">(continues on next page)</div>

```
ccc_mprun -n $((BRIDGE_MSUB_NPROC*2)) -E'--cpu_bind=threads --overcommit' ./a.out   # 4␣
→processes are run on 2 physical cores
```

- Doubling the threads:

```
#!/bin/bash
#MSUB -q <partition>
#MSUB -n 2
#MSUB -c 8
#MSUB -A <project>

export OMP_NUM_THREADS=$((BRIDGE_MSUB_NCORE*2))   # threads number must be equal to␣
→logical cores
ccc_mprun ./a.out    # each process runs 16 threads on 8 physical cores / 16 logical cores
```

17.6 Turbo

The processor turbo technology (Intel Turbo Boost or AMD turbo core) is available and activated by default on Irene. This technology dynamically adjusts CPU frequency in order to reach the highest frequency allowed by the available power and thermal limits. The Turbo can be responsible of performance variation because of hardware differences between two nodes and if other jobs are running on the same node. One can choose to disable this technology by loading the module feature/system/turbo_off:

```
module load feature/system/turbo_off
```

Deactivating this technology allows a better control of the timing of the execution of the codes but can trigger poor performances. One can choose to have capped performances by loading the feature feature/system/turbo_capped:

```
module load feature/system/turbo_capped
```

which sets the CPU frequency to a determined value which prevents Turbo to be activated for vectorized operations.

PARALLEL IO

18.1 MPI-IO

MPI-IO provides a low level implementation of parallel I/O. It was introduced as a standard in MPI-2.

More information on how to use MPI-IO can be found in the I/O section of the official MPI documentation.

18.2 Recommended data usage on parallel file system

18.2.1 Technical considerations

On the computing center, parallel file systems are implemented with Lustre technology as follows:

- One MDS stores namespace metadata, such as file names, directories, access permissions, and file layout.
- Multiple OSSes (Object Storage Servers) store file data.
- Each supercomputer node is a *client* that accesses and uses the data.

When a file is opened or created:

1. Supercomputer node requests it to the MDS.
2. The MDS performs a hand-shake between the supercomputer node and the OSSes.
3. The node can now read and/or write directly by transferring data to the OSSes.

The important point is that **all open, close and file name related calls from all the supercomputer nodes go to a unique server: the MDS**.

These calls are often referred to as *meta-data calls* and too many of these calls can badly affect the overall performances of the file system.

The following sections provide data usage guidelines to ensure the good operation of the parallel file systems.

18.2.2 Big files vs small files

Lustre file system is optimized for bandwidth rather than latency.

- While manipulating small files, I/O performances are mainly latency-based (the hand-shakes with the MDS).

- While manipulating big files, latency impact is considerably reduced.

Thus, prefer having several big files instead of having many small files. To perform basic tasks on large files or file trees, like copy, archiving and deletion, it is recommended to use MpiFileUtils as described in the *MpiFileUtils section*. These utilitaries are only effective on Lustre file systems (STORE, SCRATCH et WORK) and should not be used from or toward NFS file systems (HOME).

18.2.3 Number of files per directory

Many meta-data operations involve the MDS but also the OSSes.

Let us see with an example, a classic `ls -l`, what happens behind the scene:

1. Client asks the MDS to list all files in the current directory.

2. The MDS lists the names of the files (sub-directories, regular files, etc.).

3. The MDS knows everything about the sub-directories.

4. But MDS does not know the sizes of non-directory files (nor the date of last modification)!

5. So for each file the MDS synchronizes with each OSSes involved (sum of sizes, max of the date of modification, etc.).

Hence the more non-directory files you have within the directory, the slower the operation will be.

A similar scheme occurs to any file operation using `stat` system call for each file, like done with `find` or `du` commands.

This is the main reason why many file operations become slower as the number of regular files within a directory increases.

- As a consequence, it is strongly advised to **have less than 10.000 files per directory**.

- If you have more files, organize them into sub-directories (directories do not involve OSSes).

18.2.4 Files on STORE

STORE is a Lustre file systems linked to a Hierarchical Storage Management (HSM) relying on a magnetic tape storage system.

Thus they share the constraints of Lustre file systems and those of the magnetic tape storage.

Because tape storage has a long latency and the number of tape readers are limited, having big files becomes nearly mandatory.

That's why in order to reduce the time of retrieval :

- **each file should be bigger than 1GB** (if needed, use `tar` to aggregate your small files)

- **each file should be less than 1TB**

When you can manipulate the file size (e.g. using tarball), making it **around 100GB** is recommended. Practical experience shows that a 100GB file is quick to retrieve with `ccc_hsm get` and subsequent manipulation or extraction is also fast.

You may use `ccc_quota` to check file size statistics on STORE (they are updated daily).

18.2.5 Example of a good usage of filesystems

You have compiled your own program and installed it in your home directory under ~/products. In your home, your program will be safe. Also, you may use some user defined modules that you would create in ~/products/modules/modulefiles as described in *Extend your environment with modulefiles*. It may simplify the use of your code. To avoid reaching the quota limit on the home, don't use it by default for any work. The best place to write submission files and to launch your jobs would be the workdir. One of the advantages of the workdir is that it is shared between calculators. By launching your jobs from the workdir, you will be able to keep the submission scripts and the log files. Let us say the code generates lots of files, for example restart files, so that a new job can start where the last one stopped. In that case, the best is to launch the code on the scratchdir. It is the fastest and largest filesystem. However, data located on the scratchdir may be purged if they are not used for a while. Therefore, once your job ran successfully and you obtained the result files, you need to move them to an appropriate location if you want to keep them. That is the main purpose of the storedir. Once the job is finished, you can gather its result files in a big tar file that you shall copy on the storedir.

So, a typical job would be launched from the workdir, where the submission scripts are located. The job would go through the following steps :

- Create a temporary directory on the scratchdir and go to this directory

- Untar large input files from the storedir to our directory in the scratchdir

- Launch the code

- Once it has finished running, check the results

- If the results are not needed at the moment for other runs, make a tar of the useful files and copy the tar on the storedir

- Remove the temporary directory

Note that those steps are not compulsory and do not apply to every job. It is just one possible example of correct use of the different filesystems.

18.3 Parallel compression and decompression with pigz

Instead of using classical compression tools like **gzip** (included by default with **tar**), we recommend to use his parallel counterpart **pigz** to get faster processing.

With this tool, we advise you to limit the number of threads used for compression between 4 and 8 to have a good performance / resources ratio. Increasing the number of threads should not dramatically improve performance and could even slow your compression. To speed up the process you may also adjust the compression level at the cost of reducing the compression quality. Decompression will be done by only one thread in any case and three more threads will be used for various purposes (read, write, check).

Please do not use this tool on login nodes, and prefer an interactive submission with **ccc_mprun -s** or with a batch script.

Compression and decompression example using 6 threads:

```bash
#!/bin/bash
#MSUB -n 1
#MSUB -c 6
#MSUB -q <partition>
#MSUB -A <Project>
module load pigz
```

```
# compression:
# we are forced to create a wrapper around pigz if we want to use
# specific options to change the default behaviour of pigz.
# Note that '$@' is important because tar can pass arguments to pigz
cat <<EOF > pigz.sh
#!/bin/bash
pigz -p 6 \$@
EOF
chmod +x ./pigz.sh

tar -I ./pigz.sh -cf folder.tar.gz folder

# decompression:
tar -I pigz -xf folder.tar.gz
```

For additional information, please refer to the man-pages of the software:

```
$ man pigz
```

18.4 MpiFileUtils

MpiFileUtils is a suite of utilitaries allowing for handling file trees and large files. It is optimised for HPC and use MPI paralellisation. It offers tools for basic tasks like copy, remove, and compare for such datasets, delivering better performance than their single-process counterparts.

dcp - Copy files Using 64 processes and 16 cores, dcp provides a data rater more than 6 times greater than a regular cp using a full node to copy 80GB/1800 files on the scratch file system.

dtar - Create and extract tape archive files Using 64 processes and 16 cores, dtar provides a data rater more than 6 times greater than a regular tar using a full node to archive 80GB/1800 files on the scratch file system.

dbcast - Broadcast a file to each compute node.

dbz2 - Compress and decompress a file with bz2.

dchmod - Change owner, group, and permissions on files.

dcmp - Compare contents between directories or files.

ddup - Find duplicate files.

dfind - Filter files.

dreln - Update symlinks to point to a new path.

drm - Remove files.

dstripe - Restripe files (Lustre).

dsync - Synchronize source and destination directories or files.

dtar - Create and extract tape archive files.

dwalk - List, sort, and profile files.

Sample script for dcp, dtar and drm

```
#!/bin/bash
#MSUB -r dcp_dtar
#MSUB -q |default_CPU_partition|
#MSUB -T 3600
#MSUB -n 64
#MSUB -c 16
#MSUB -x
#MSUB -A <Project>
#MSUB -m work,scratch
ml pu
ml mpi
ml mpifileutils
ccc_mprun dtar -cf to_copy.tar to_tar/ # Create an archive
ccc_mprun dcp to_copy.tar copy_dcp.tar # Copy the archive
ccc_mprun drm copy_dcp.tar # Remove the copy of the archive
```

Sample script for dstripe

```
#!/bin/bash
#MSUB -r dstripe
#MSUB -q |default_CPU_partition|
#MSUB -T 1500
#MSUB -Q test
#MSUB -n 16
#MSUB -c 8
#MSUB -x
#MSUB -A <Project>
#MSUB -m work,scratch
ml gnu/8
ml mpifileutils
ccc_mprun dstripe -c <number of desired stripe> to_copy.tar
```

For more information, see: https://mpifileutils.readthedocs.io/en/v0.11.1/

DEBUGGING

Parallel applications are difficult to debug. Depending on the kind of problem, the type of parallelism, some tools may provide a great help in the debugging process.

19.1 Summary

Table 1: Supported programming model and functionality

Name	MPI	OpenMP	Cuda	GUI	Step by step	Memory Debugging
Linaro-forge DDT	✓	✓	✓	✓	✓	✓
GDB					✓	
PDB					✓	
Intel Inspector		✓		✓		✓
Totalview	✓	✓	✓	✓	✓	✓
Valgrind	✓					✓

To display a list of all available debuggers use the search option of the module command:

```
$ module search debugger
```

19.2 Compiler flags

19.2.1 Common flags

To debug codes, you need to enable debug symbols. You get these symbols by compiling with the appropriate options:

- `-g` to generate debug symbols usable by most debugging and profiling tools.

- or `-g3` to generate even more debugging information (available for GNU and Intel, C, C++ or Fortran compilers).

- and optionally `-O0` to avoid code optimization (this is strongly recommended for first debug sessions).

19.2.2 Flags for Fortran

- `-traceback` with **ifort** or `-fbacktrace` with **gfortran**: specifies that a backtrace should be produced if the program crashes, showing which functions or subroutines were being called when the error occurs.

For example, when getting a segmentation fault in Fortran, you may get the following error message which is not very useful:

```
forrtl: severe (174): SIGSEGV, segmentation fault occurred
Image      PC                Routine Line    Source
run_exe  000000010005EAC7 Unknown Unknown Unknown
run_exe  000000010005DDA9 Unknown Unknown Unknown
run_exe  00000001000009BC Unknown Unknown Unknown
run_exe  0000000100000954 Unknown Unknown Unknown
```

A code compiled with `-fbacktrace` or `-traceback` will give a more relevant output:

```
forrtl: severe (174): SIGSEGV, segmentation fault occurred
Image      PC                Routine Line    Source
run_exe  000000010005EAC7 test_m_   265     mod_test.f90
run_exe  000000010005DDA9 io_       52      io.f90
run_exe  00000001000009BC setup_    65      test_Setup.f90
run_exe  0000000100000954 main_     110     launch.f90
```

- `-check bounds` with **ifort** or `-fbounds-check` with **gfortran**: checks that an index is within the bounds of the array each time an array element is accessed. This option is expected to substantially slow down program execution but is a convenient way to track down bugs related to arrays. Without this flag, an illegal array access would produce either a subtle error that might not become apparent until much later in the program or will cause an immediate segmentation fault with poor information on the origin of the error.

Note: Be careful. Most of these compiler options will slow down your code performances.

19.3 GDB

GDB is the Gnu DeBugger. It is a lightweight simple serial debugger available on most systems.

To start a program under GDB, first make sure it is compiled with `-g`. Start a GDB session for your code:

```
$ gdb ./gdb_test
GNU gdb (GDB) Red Hat Enterprise Linux
Copyright (C) 2010 Free Software Foundation, Inc.
(gdb)
```

Once the GDB session is started, launch the code with:

```
(gdb) run
```

If an error occurs, you will be able to get information with **backtrace**:

```
Program received signal SIGSEGV, Segmentation fault.
(gdb) backtrace
```

```
#0  0x00000000004005e0 in func1 (rank=1) at test.c:14
#1  0x0000000000400667 in main (argc=1, argv=0x7fffffffacc8) at test.c:30
```

GDB allows to set breakpoints, run the code step by step and more. See **man gdb** for more information and options.

GDB can be used on one process at a time with a parallel program. To attach GDB to a running process you may use the following method :

- Compile the program with debug options.

- Start the program

- Find on wich nodes the program is running using the **ccc_mpp -u $USER** command

- Connect to a compute node used by the program, using the **ssh <compute node>** command

- Find the process ID of your application using the **ps -fu** command

- Connect to a running process using the **gdb -p <process id>** command

You can use gdb on several processes at the same time.

19.4 DDT

DDT is a highly scalable debugger specifically adapted to supercomputers.

19.4.1 Basics

You can use DDT after loading the appropriate module:

```
$ module load linaro-forge
```

Note: Allinea-forge has been renamed Arm-forge, which has then been renamed Linaro-forge.

Then use the command **ddt**. For parallel codes, edit your submission script and replace the line

```
$ ccc_mprun -n 16 ./a.out
```

with:

```
$ ddt -n 16 ./a.out
```

You may want to add the -noqueue option to make sure DDT will not submit a new job to the scheduler. You have to specify the good version of the mpi distribution by selecting **run** and select **SLURM (generic)** implementation as shown on the figures below.

Example of submission script:

```
$ cat ddt.job
#!/bin/bash
#MSUB -r MyJob_Para       # Job name
#MSUB -q <partition>      # Partition name
#MSUB -A <project>        # Project ID
```
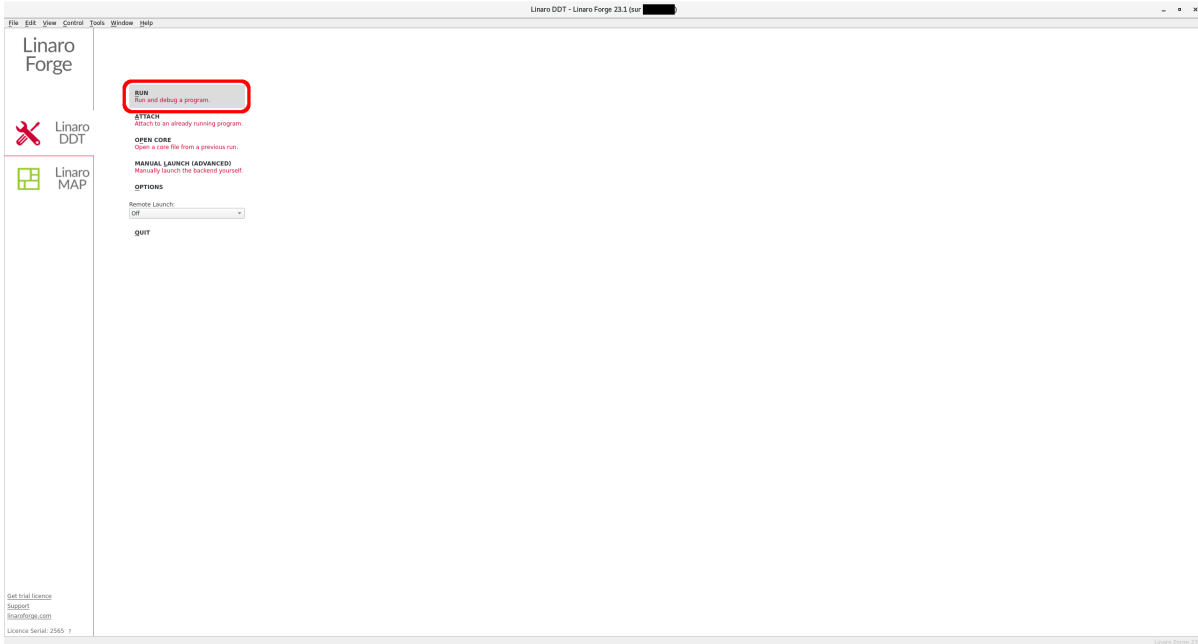
Fig. 1: DDT opening window: choose 'Run'

```
#MSUB -n 32              # Number of tasks to use
#MSUB -T 1800            # Elapsed time limit in seconds
#MSUB -o example_%I.o    # Standard output. %I is the job id
#MSUB -e example_%I.e    # Error output. %I is the job id
set -x
cd ${BRIDGE_MSUB_PWD}
ddt -n 32 ./ddt_test


$ ccc_msub -X ddt.job
```

Note: The `-X` option for **ccc_msub** enables X11 forwarding.

19.4.2 DDT with NiceDCV

If debugging with DDT requires more performance than what can provide the X11 forwarding, you may use NiceDCV. First, start ddt on NiceDCV.

```
$ module load linaro-forge
$ ddt
```

Then select Manual Launch and indicate the number of processes:

Then submit your code using the **ddt-client** command :

```
cat submit_visu.sh
```
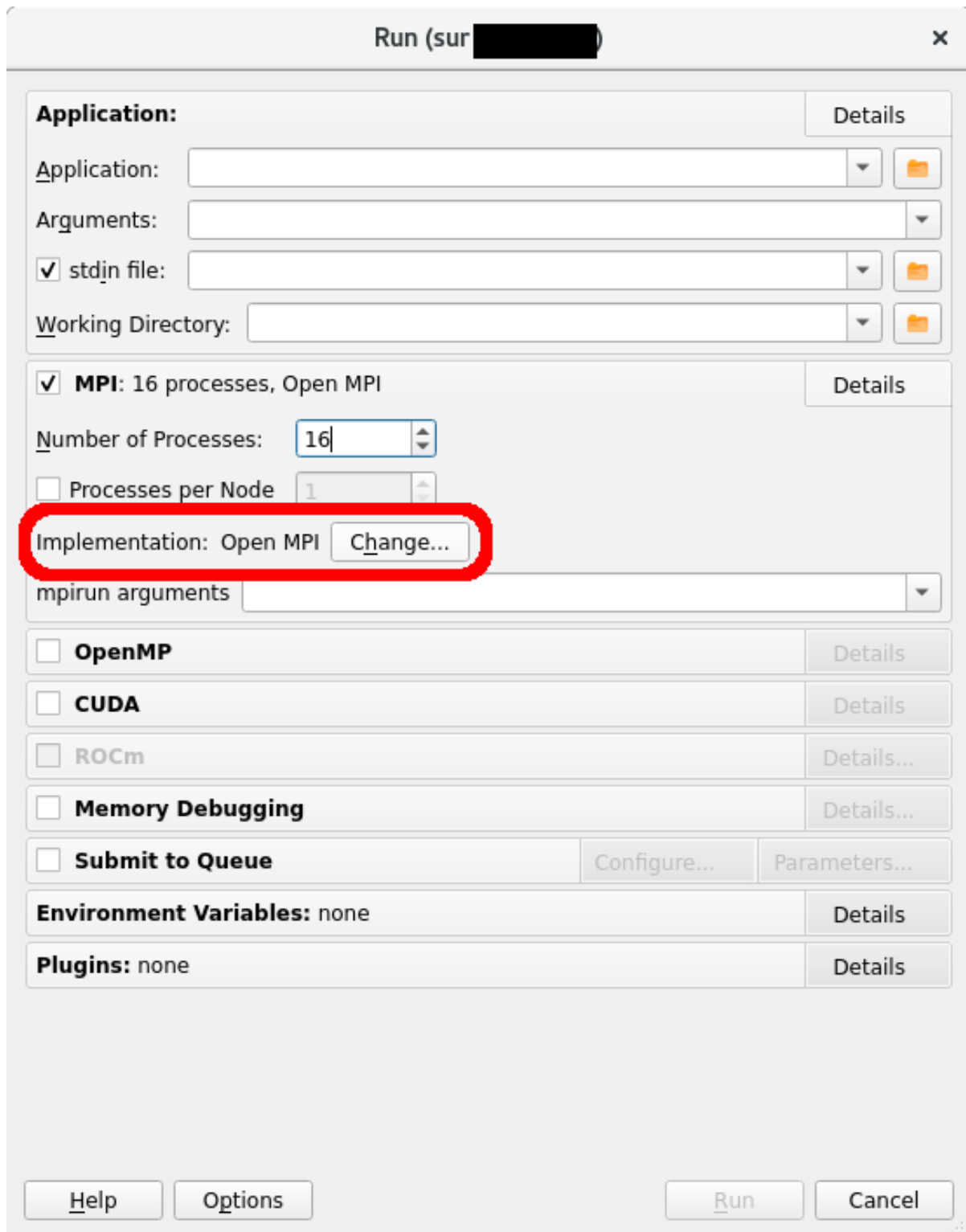
Fig. 2: Choose 'change'
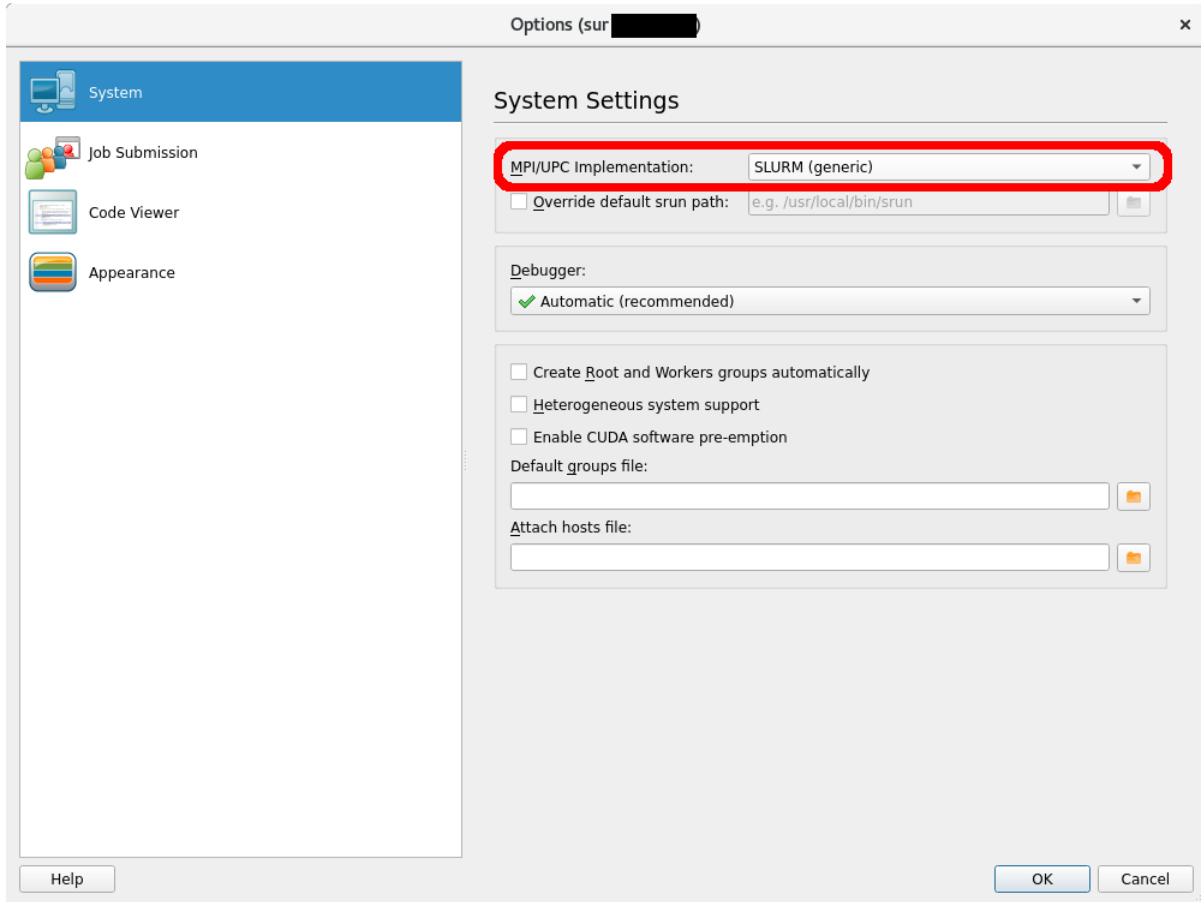
Fig. 3: Choose 'SLURM (generic)'
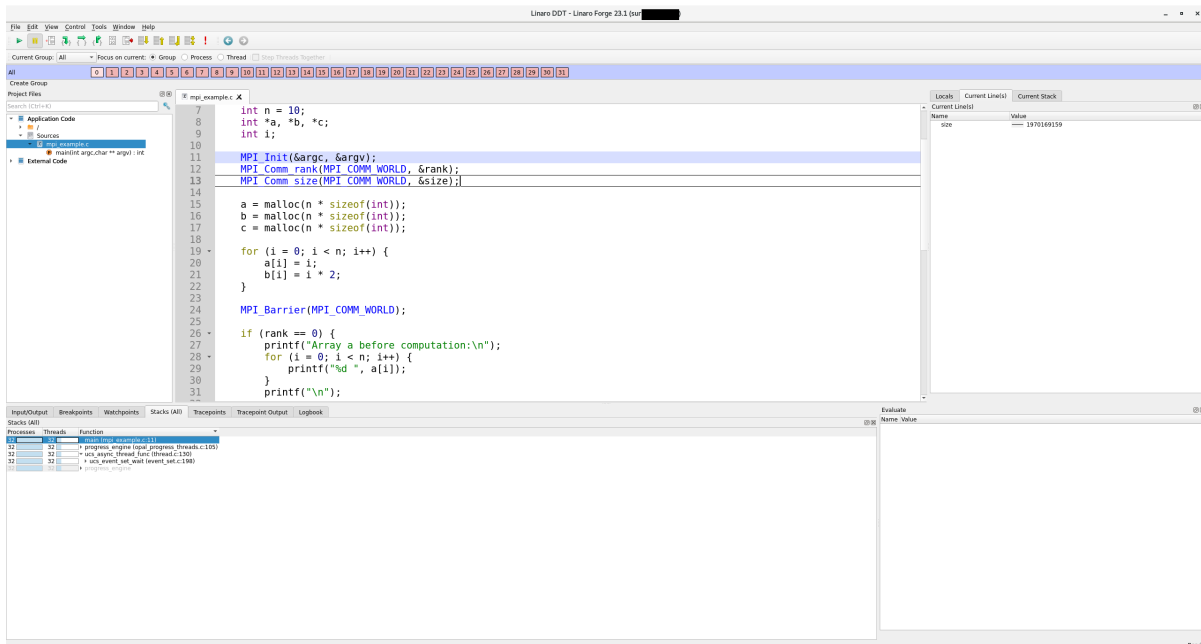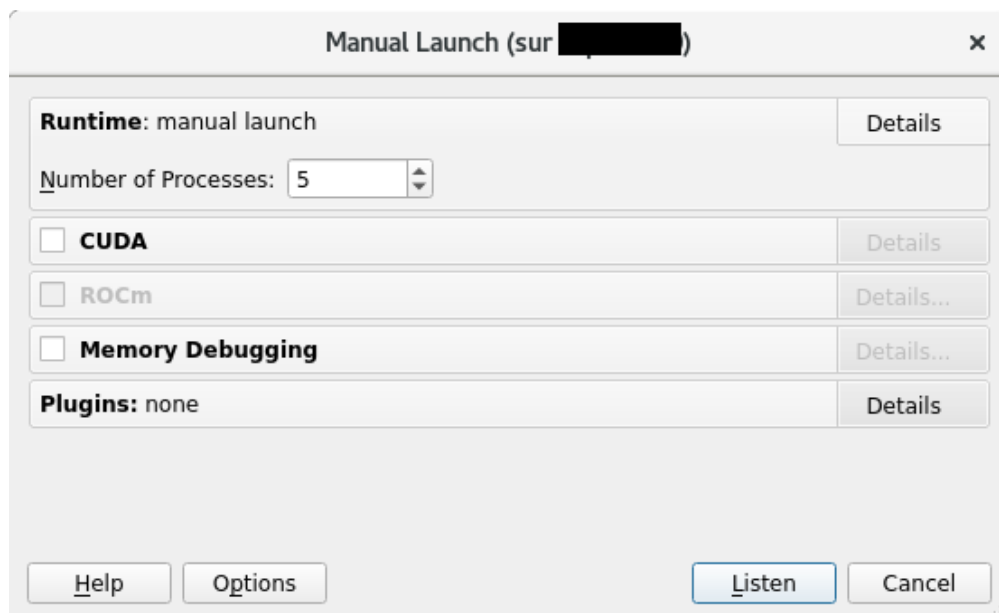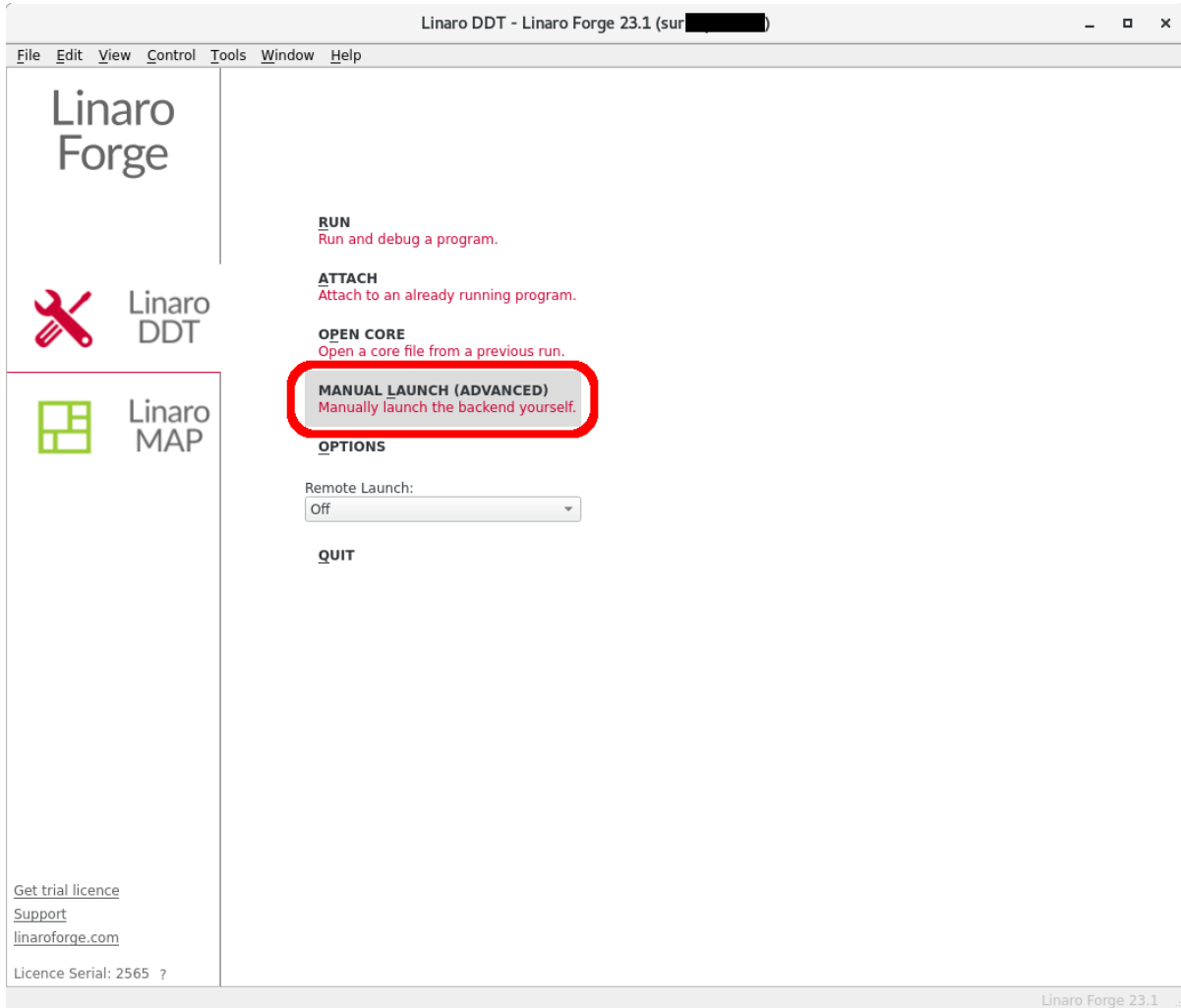
```
#!/bin/bash
#MSUB -r TP4_debugging
#MSUB -n 16
#MSUB -T 1800
#MSUB -q <partition>
#MSUB -A <project>
#MSUB -m work,scratch
#MSUB -e TP4_debugging_%J.err
#MSUB -o TP4_debugging_%J.out


ml purge
ml mpi


ml linaro-forge
ccc_mprun ddt-client  ./cstartmpi
```

DDT should be able to catch the launch and you may use DDT as usual.

Note: Linaro-forge DDT is a licensed product.

A full documentation is available in the installation path on the cluster. To open it:

```
$ evince ${LINAROFORGE_ROOT}/doc/userguide-forge.pdf
```

19.4.3 Advanced: debug MPMD scripts

Prior to start **ddt** you need to create an appropriate script in MPMD mode:

```
$ cat ddt.job
#!/bin/bash
#MSUB -r MyJob_Para        # Job name
#MSUB -q <partition>       # Partition name
#MSUB -A <project>         # Project ID
#MSUB -n 4                 # Number of tasks to use
#MSUB -T 1800              # Elapsed time limit in seconds
#MSUB -X
#MSUB -o example_%I.o      # Standard output. %I is the job id
#MSUB -e example_%I.e      # Error output. %I is the job id
set -x
cd ${BRIDGE_MSUB_PWD}


module load linaro-forge
cat << END > exe.conf
1   env ddt-client ./algo1
3   env ddt-client ./algo2
END


ccc_mprun -f exe.conf
```

Now, as well as before, load the appropriate module:

```
$ module load linaro-forge
```

Then start **ddt**:

```
$ ddt&
```

Once ddt interface is visible, select `MANUAL LAUNCH`:
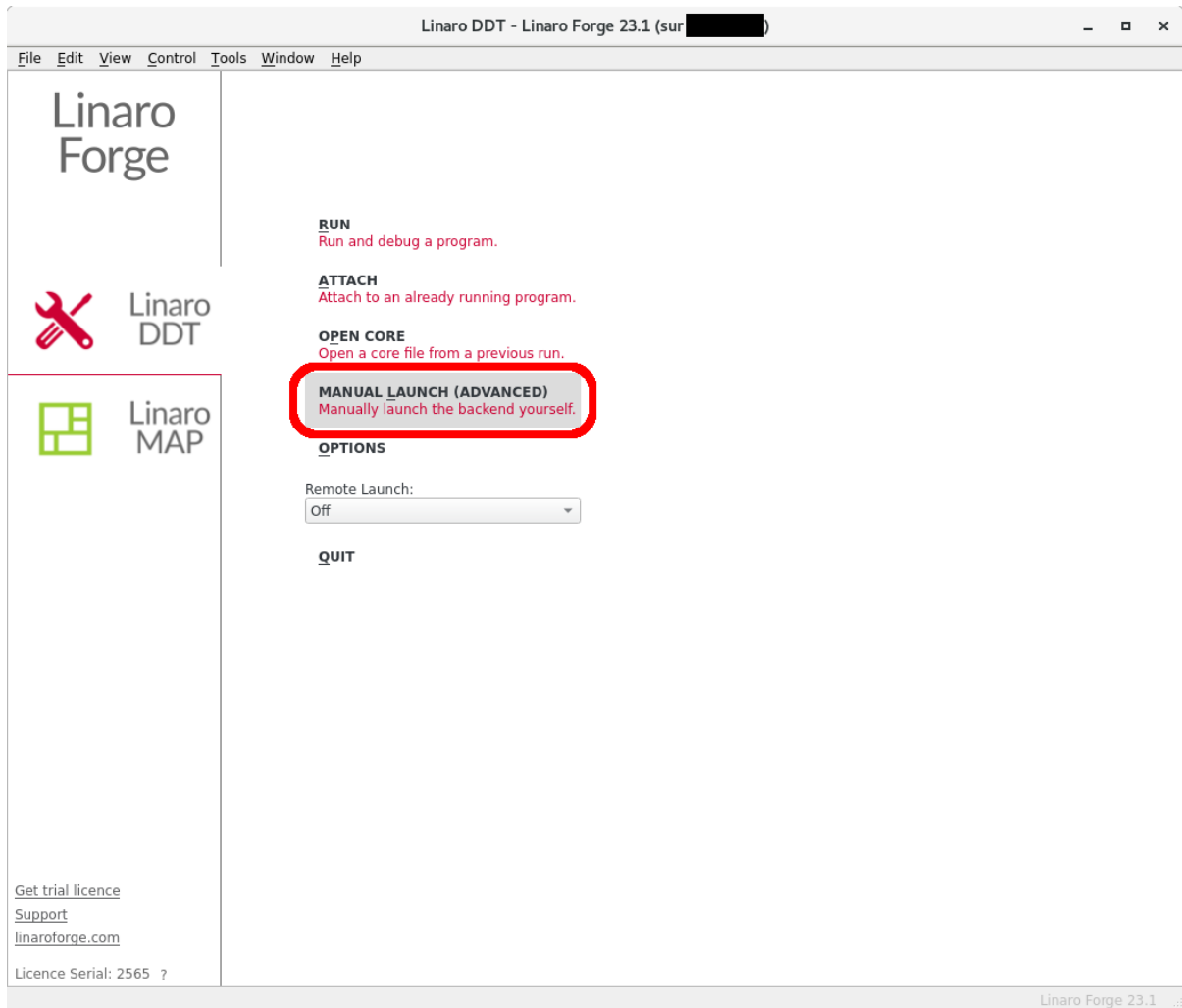
Select the same number of processes that you choose on your script at `#MSUB -n` (4 here) and press `Listen`:
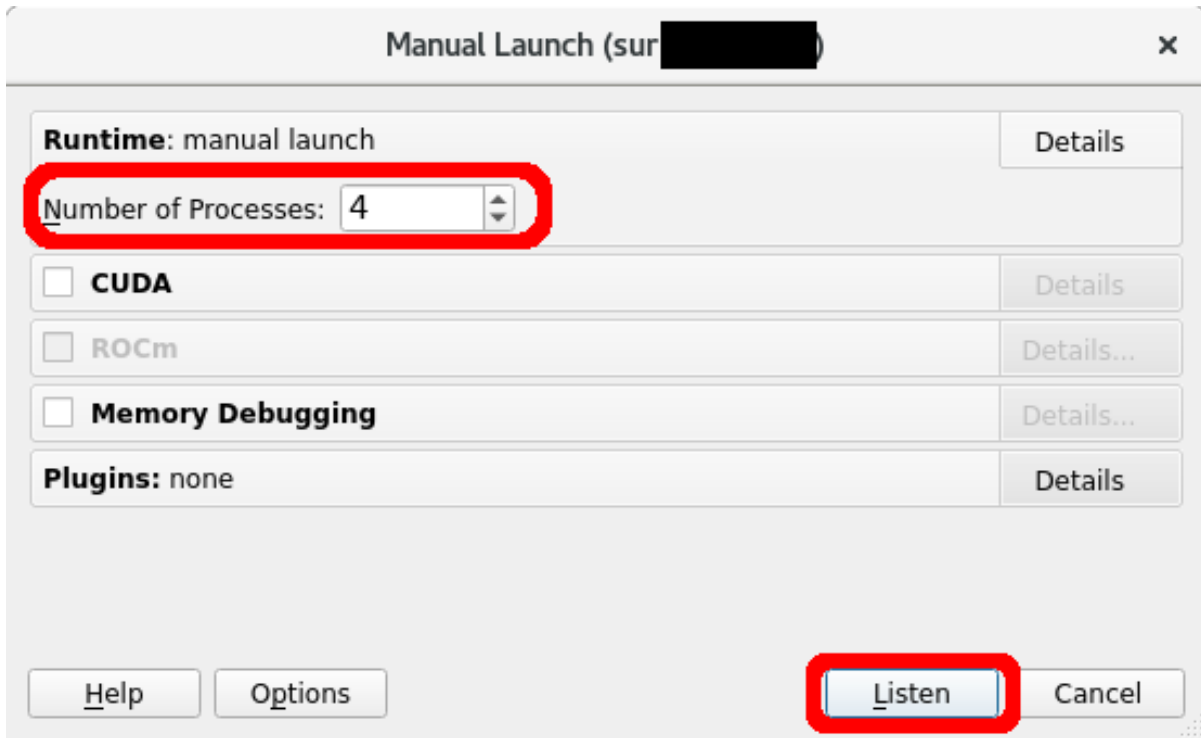
Then, launch your script:

```
$ ccc_msub -X ddt.job
```

Wait and your job will be automatically attach to **ddt**. Now you have an an interface with `algo1` and `algo2` running at the same time:
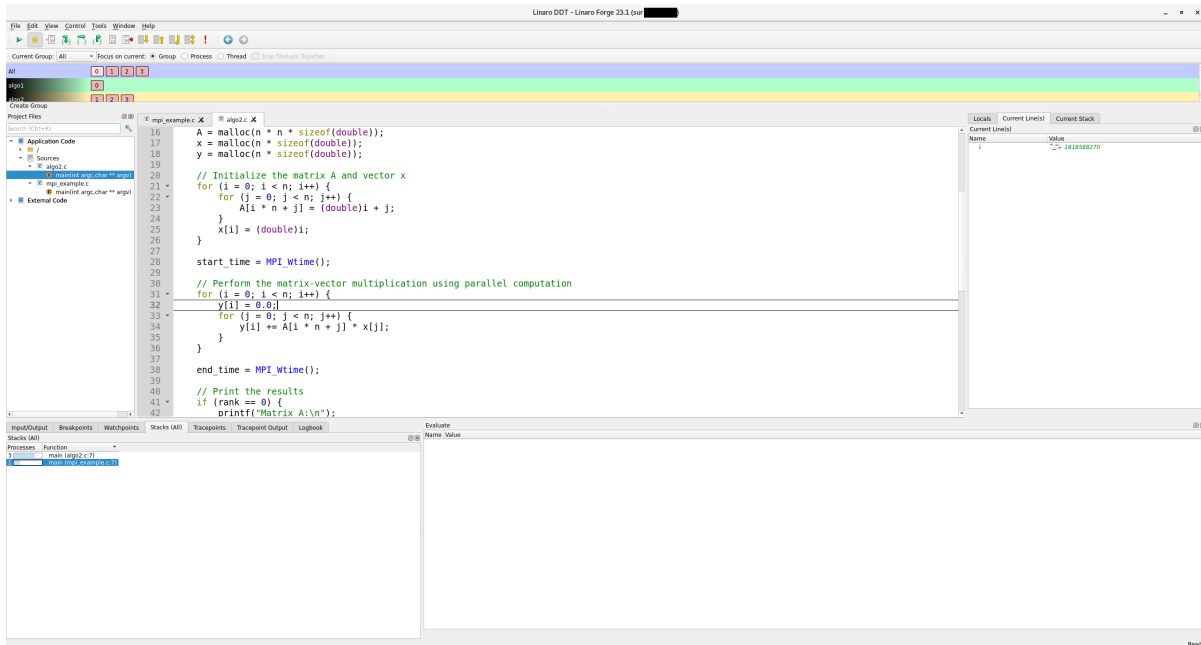
19.5 TotalView

TotalView may be used by loading a module and by submitting an appropriate job:

```
$ module load totalview
```

Then launch your job with a submission script like:

```
#!/bin/bash
#MSUB -r MyJob            # Job name
#MSUB -q <partition>      # Partition name
#MSUB -A <project>        # Project ID
#MSUB -n 8                # Number of tasks to use
#MSUB -T 600              # Time limit
#MSUB -o totalview_%I.o   # Standard output. %I is the job id
#MSUB -e totalview_%I.e   # Error output. %I is the job id
set -x
cd ${BRIDGE_MSUB_PWD}
ccc_mprun -d tv ./totalview_test
```

It needs to be submitted with:

```
$ ccc_msub -X totalview.job
```

Totalview should open on the Startup Parameters window. There is nothing to change here, just hit **OK**. Once in the main window, you can either come back to the parameter window with "<ctrl-a>" or launch the code with "g".

Note: Totalview is a licensed product.

Check the output of `module show totalview` or `module help totalview` to get more information on the amount of licenses available.

A full documentation is available in the installation path on the cluster. To open it:

```
$ evince ${TOTALVIEW_ROOT}/doc/pdf/TotalView_User_Guide.pdf
```

19.6 Pdb Python debugger

pdb is a built-in Python debugger that aids in inspecting your code, setting breakpoints, and understanding program flow.

First of all, load *python3* module:

```
$ module load python3
```

To start pdb when running your script, use the python command *-m pdb* option:

```
$ python3 -m pdb monte_carlo_pi.py
> monte_carlo_pi.py(1)<module>()
-> import random
(Pdb) ...
```
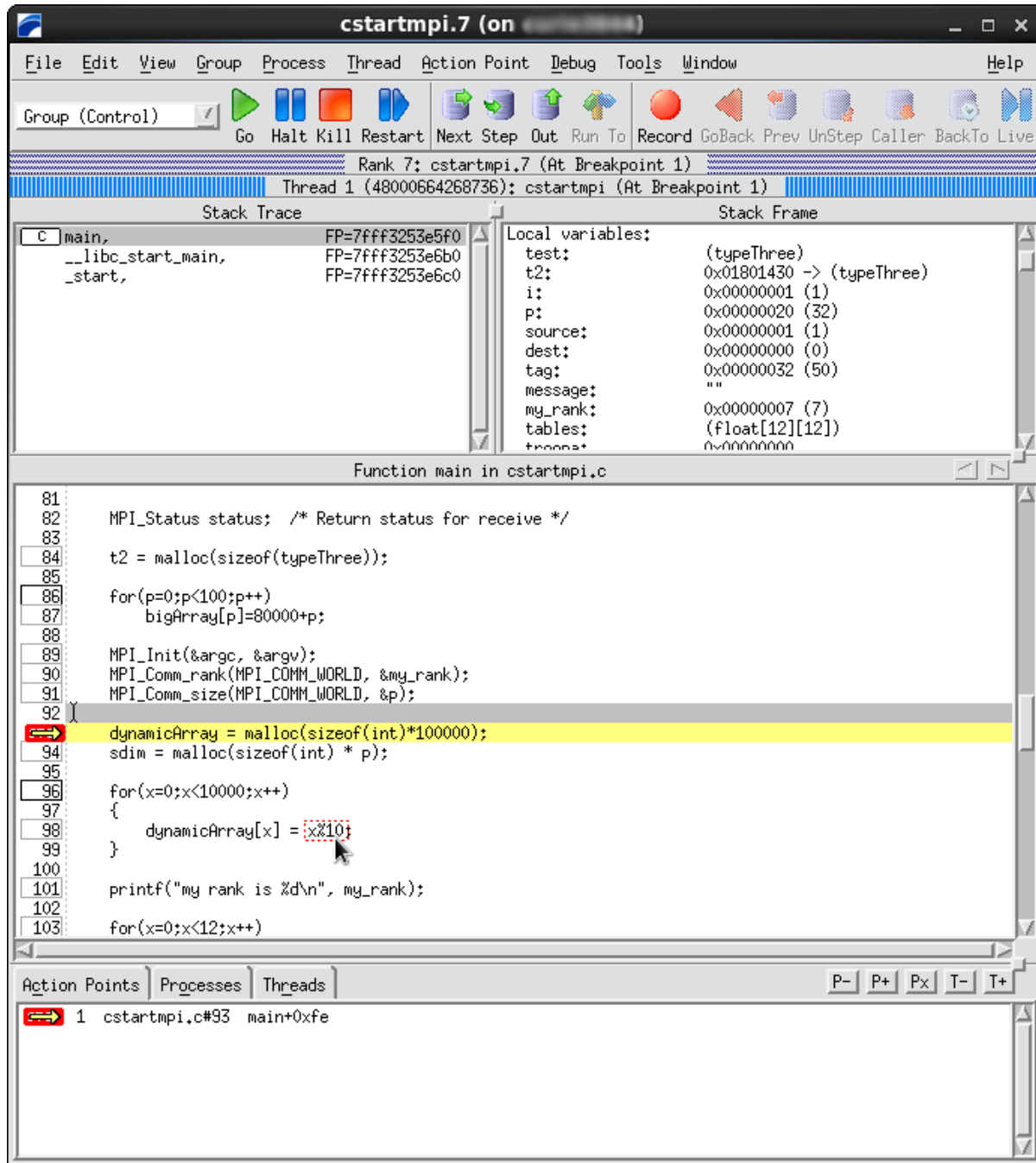
Fig. 4: Example of Totalview window

A prompt is opened, use *help* command:

```
(Pdb) help

Documented commands (type help <topic>):
========================================
EOF    c         d        h         list      q        rv       undisplay
a      cl        debug    help      ll        quit     s        unt
alias  clear     disable  ignore    longlist  r        source   until
args   commands  display  interact  n         restart  step     up
b      condition down     j         next      return   tbreak   w
break  cont      enable   jump      p         retval   u        whatis
bt     continue  exit     l         pp        run      unalias  where
```

Use *help <topic>* for more information about any command:

```
(Pdb) help a
a(rgs)
      Print the argument list of the current function.
```

Use a breakpoint (*break <file>:<line>*) to stop the code when a file line is reached:

```
(Pdb) break monte_carlo_pi.py:10
Breakpoint 1 at monte_carlo_pi.py:10
(Pdb) continue
> monte_carlo_pi.py(10)estimate_pi()
-> distance = x**2 + y**2
```

Use *list .* command to list the current code:

```
(Pdb) list .
  3         def estimate_pi(num_points):
  4             points_in_circle = 0
  5
  6             for _ in range(num_points):
  7                 x = random.uniform(0, 1)
  8                 y = random.uniform(0, 1)
  9
 10 B->             distance = x**2 + y**2
 11             if distance <= 1:
 12                 points_in_circle += 1
 13
 14             return 4 * points_in_circle / num_points
```

Note the *B* for *Breakpoint* and *->* indicates the current line.

Pdb is able to print values with *p* or *pp* commands:

```
(Pdb) p distance
*** NameError: name 'distance' is not defined
```

If you are looking for any local variable, use *locals()* function:

```
(Pdb) locals()
{'num_points': 10000000, 'points_in_circle': 0, '_': 0, 'x': 0.2584912119080409, 'y': 0.
```

(continues on next page)

```
→6628071583040221}
```

Note there is a equivalent for *globals()* but could be very long.

With the *next* command, execute the code line per line:

```
(Pdb) next
> /ccc/work/cont000/asplus/cotte/cProfile/monte_carlo_pi.py(11)estimate_pi()
-> if distance <= 1:
(Pdb) p distance
0.5061310357327408
(Pdb) ll
  3          def estimate_pi(num_points):
  4              points_in_circle = 0
  5
  6              for _ in range(num_points):
  7                  x = random.uniform(0, 1)
  8                  y = random.uniform(0, 1)
  9
 10 B                distance = x**2 + y**2
 11  ->             if distance <= 1:
 12                      points_in_circle += 1
 13
 14              return 4 * points_in_circle / num_points
```

Note the -> has moved and *distance* is defined now.

Here are the main usefull commands:

Command	Description
b(reak)	Set a breakpoint at specified line number or function, with an optional condition.
c(ont)	Continue execution, only stop when a breakpoint is encountered.
l(ist)	Displays 11 lines around the current line (*l .*) or continue the previous listing.
ll	List the whole source code for the current function or frame.
p / pp	Evaluate and print the expression in Python syntax. Use *pp* for tables/structures.
locals()	Return a dictionary of the current namespace.
globals()	Return a dictionary of the current global namespace.
s(tep)	Execute the current line, stop at the first possible occasion.
n(ext)	Continue execution until the next line in the current function is reached or it returns.
r(eturn)	Continue execution until the current function returns.
q(uit)	Quit from the debugger.

For more information, please refer to the official Python documentation.

19.7 Other tools

19.7.1 Valgrind Memcheck

Valgrind is an instrumentation framework for dynamic analysis tools. It comes with a set of tools for profiling and debugging.

Memcheck is a memory error detector. It is the default use of Valgrind so any call to **valgrind** is equivalent to calling

```
$ valgrind --tools=memcheck
```

To check your code with Valgrind, just call **valgrind** before the program :

```
$ module load valgrind
$ valgrind ./test
```

To run MPI programs under Valgrind, use the available library "libmpiwrap" to filter false positives on MPI functions. It is available through the VALGRIND_PRELOAD environment variable. It is also possible to specify the output file and to force Valgrind to output one file per process (with --log-file).

```bash
#!/bin/bash
#MSUB -n 32
#MSUB -T 1800
#MSUB -q <partition>
#MSUB -A <project>

module load valgrind

export LD_PRELOAD=${VALGRIND_PRELOAD}

ccc_mprun valgrind --log-file=valgrind_%q{SLURM_JOBID}_%q{SLURM_PROCID} ./test
```

Here is the kind of output Valgrind returns :

```
==22860== Invalid write of size 4
==22860== at 0x4005DD: func1 (test1.c:12)
==22860== by 0x40061E: main (test1.c:20)
==22860== Address 0x4c11068 is 0 bytes after a block of size 40 alloc'd
==22860== at 0x4A05FDE: malloc (vg_replace_malloc.c:236)
==22860== by 0x4005B0: func1(test1.c:9)
==22860== by 0x40061E: main (test1.c:20)
```

PROFILING

20.1 Summary

Table 1: Supported programming model

Name	MPI	OpenMP	Cuda	SIMD
Advisor		✓		✓
AMD µProf	✓	✓		
Linaro MAP	✓	✓	✓	✓
Cube	✓	✓	✓	✓
Darshan	✓	✓		
Gprof				
cProfile				
HPCToolkit	✓	✓		
Igprof				
IPM	✓			
Memonit	✓	✓		
SelFle				
Paraver	✓	✓	✓	
PAPI				✓
Perf				✓
ScoreP	✓	✓	✓	
Tau	✓	✓	✓	
Valgrind (cachegrind)	✓	✓		
Valgrind (callgrind)	✓	✓		
Valgrind (massif)	✓	✓		
Vampir	✓	✓	✓	
Vtune		✓		✓

Table 2: Collectible events

Name	Comm	I/O	Call graph	Hardware counters	Memory usage	Cache usage
Advisor			✓			
AMD µProf				✓	✓	✓
Linaro MAP	✓	✓			✓	
Cube	✓	✓			✓	
Darshan		✓				
Gprof			✓			
cProfile			✓			
HPCToolkit		✓	✓	✓		
Igprof					✓	
IPM	✓			✓		
Memonit					✓	
SelFIe	✓	✓	✓			✓
Paraver	✓			✓		
PAPI				✓	✓	✓
Perf		✓	✓	✓	✓	✓
ScoreP	✓		✓	✓	✓	✓
Tau	✓	✓	✓		✓	
Valgrind (cachegrind)						✓
Valgrind (callgrind)			✓			
Valgrind (massif)					✓	
Vampir	✓					
Vtune				✓		✓

Table 3: Tool support and type of profiling

Name	Collection	GUI	Sampling	Tracing	Instrumentation necessary
Advisor	✓	✓	✓		
AMD µProf	✓	✓	✓	✓	
Linaro MAP	✓	✓	✓	✓	
Cube	✓	✓	✓	✓	
Darshan	✓		✓		
Gprof	✓		✓	✓	✓
cProfile	✓		✓	✓	✓
HPCToolkit	✓	✓	✓	✓	
Igprof	✓		✓	✓	
IPM	✓		✓		
Memonit	✓	✓	✓		
SelFIe	✓		✓		
Paraver		✓		✓	✓
PAPI	✓		✓		✓
Perf			✓		
ScoreP	✓		✓	✓	✓
Tau	✓	✓	✓	✓	
Valgrind (cachegrind)	✓		✓		
Valgrind (callgrind)	✓		✓		
Valgrind (massif)	✓		✓		
Vampir		✓		✓	✓
Vtune	✓	✓	✓		

To display a list of all available profilers use the `search` option of the **module** command :

```
$ module search profiler
```

20.2 SelFIe

SelFIe (SElf and Light proFIling Engine) is a tool to lightly profile Linux commands without compiling. The profiling is done by a dynamic library which can be given to the LD_PRELOAD environment variable before the execution of the command. It doesn't affect the behaviour of the command and users don't see any changes at execution. At the end of the execution, it puts a line in system logs:

Note: SelFIe is a opensource software developed by CEA - Selfie on Github

---

```
selfie[26058]: { "utime": 0.00, "stime": 0.01, "maxmem": 0.00, "posixio_time": 0.00,
↪"posixio_count": 7569, "USER": "user", "wtime": 0.01, "command": "/bin/hostname" }
```

- To enable selfie:

```
$ module load feature/selfie/enable
```

- To disable selfie:

```
$ module load feature/selfie/disable
```

- To display the log in standard output:

```
$ module load feature/selfie/report
```

- If your process takes less than 5 minutes, then load:

```
$ module load feature/selfie/short_job
```

- SelFIe data can be written in a custom file by exporting the environment variable `SELFIE_OUTPUTFILE`:

```
$ export SELFIE_OUTPUTFILE=selfie.data
$ ccc_mprun -p |default_CPU_partition| -Q test -T 300 ./a.out
$ cat selfie.data
selfie[2152192]: { "utime": 0.00, "stime": 0.00, "maxmem": 0.01, "hostname": "host1216",
↪"posixio_time": 0.00, "posixio_count": 4, "mpi_time": 0.00, "mpi_count": 0, "mpiio_time
↪": 0.00, "mpiio_count": 0, "USER": "username", "SLURM_JOBID": "3492220", "SLURM_STEPID
↪": "0", "SLURM_PROCID": "0", "OMP_NUM_THREADS": "1", "timestamp": 1687937047, "wtime":␣
↪30.00, "command": "./a.out" }
```

- Example of a short job in standard output:

```
$ module load feature/selfie/enable
$ module load feature/selfie/report
$ module load feature/selfie/short_job
$ ccc_mprun -p |default_CPU_partition| -Q test -T 300 ./a.out

selfie[2152192]: { "utime": 0.00, "stime": 0.00, "maxmem": 0.01, "hostname": "host1216",
```

(continues on next page)

```
→"posixio_time": 0.00, "posixio_count": 4, "mpi_time": 0.00, "mpi_count": 0, "mpiio_time
→": 0.00, "mpiio_count": 0, "USER": "username", "SLURM_JOBID": "3492220", "SLURM_STEPID
→": "0", "SLURM_PROCID": "0", "OMP_NUM_THREADS": "1", "timestamp": 1687937047, "wtime":␣
→30.00, "command": "./a.out" }
```

- You can format the JSON output via the *jq* command:

```
 $ jq . <<< '{ "utime": 0.00, "stime": 0.00, "maxmem": 0.01, "hostname": "host1216",
→"posixio_time": 0.00, "posixio_count": 4, "mpi_time": 0.00, "mpi_count": 0, "mpiio_time
→": 0.00, "mpiio_count": 0, "USER": "username", "SLURM_JOBID": "3492220", "SLURM_STEPID
→": "0", "SLURM_PROCID": "0", "OMP_NUM_THREADS": "1", "timestamp": 1687937047, "wtime":␣
→30.00, "command": "./a.out" }'

{
  "utime": 0,
  "stime": 0,
  "maxmem": 0.01,
  "hostname": "host1216",
  "posixio_time": 0,
  "posixio_count": 4,
  "mpi_time": 0,
  "mpi_count": 0,
  "mpiio_time": 0,
  "mpiio_count": 0,
  "USER": "username",
  "SLURM_JOBID": "3492220",
  "SLURM_STEPID": "0",
  "SLURM_PROCID": "0",
  "OMP_NUM_THREADS": "1",
  "timestamp": 1687937047,
  "wtime": 30,
  "command": "./a.out"
}
```

20.3 IPM

IPM is a light-weight profiling tool that profiles the mpi calls and memory usage in a parallel program. IPM cannot be used on a multi-threaded program.

To run a program with IPM profiling, just load the **ipm module** (no need to instrument or recompile anything) and run it with :

```
#!/bin/bash
#MSUB -r MyJob_Para            # Job name
#MSUB -n 32                    # Number of tasks to use
#MSUB -T 1800                  # Elapsed time limit in seconds
#MSUB -o example_%I.o          # Standard output. %I is the job id
#MSUB -e example_%I.e          # Error output. %I is the job id
#MSUB -q <partition>           # Partition name
#MSUB -A <project>             # Project ID
```

```
set -x
cd ${BRIDGE_MSUB_PWD}

module load ipm

#The ipm module tells ccc_mprun to use IPM library
ccc_mprun ./prog.exe
```

It will generate a report at the end of the standard output of the job and an **xml file**. It is possible to generate a graphical and complete html page with the command:

```
$ ipm_parse -html XML_File
```

Example of IPM output

Example of IPM output
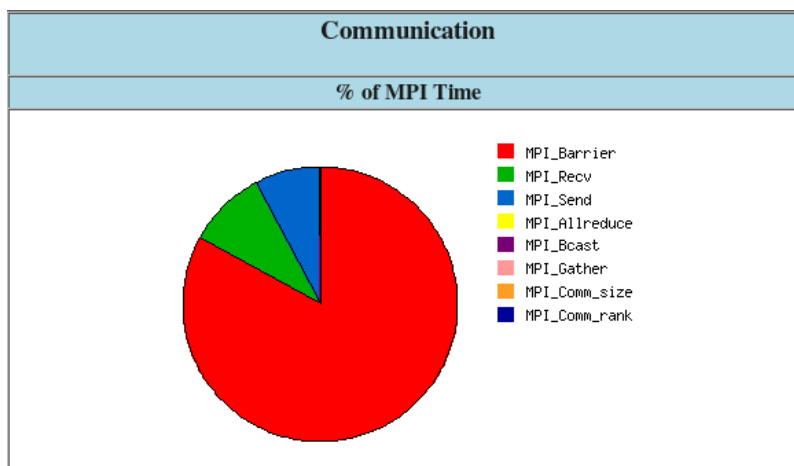
Communication Event Statistics (100.00% detail, -7.4831e-05 error)							
	Buffer Size	**Ncalls**	**Total Time**	**Min Time**	**Max Time**	**%MPI**	**%Wall**
MPI_Barrier	0	612	25.281	8.276e-03	1.588e+00	83.05	72.35
MPI_Recv	536870912	4	0.798	1.993e-01	1.996e-01	2.62	2.28
MPI_Send	536870912	4	0.797	1.991e-01	1.994e-01	2.62	2.28
MPI_Recv	268435456	4	0.400	9.987e-02	9.996e-02	1.31	1.14

20.4 Linaro-forge MAP

Linaro-forge MAP is the profiler for parallel, multithreaded or single threaded C, C++ and F90 codes. MAP gives information on memory usage, MPI and OpenMP usage, percentage of vectorized SIMD instructions, etc.

The code just has to be compiled with -g for debugging information. No instrumentation is needed.

You can profile your code with map by loading the appropriate module:

```
$ module load linaro-forge
```

Then use the command **map --profile**. For parallel codes, edit your submission script and just replace **ccc_mprun** with **map --profile**.

Example of submission script:
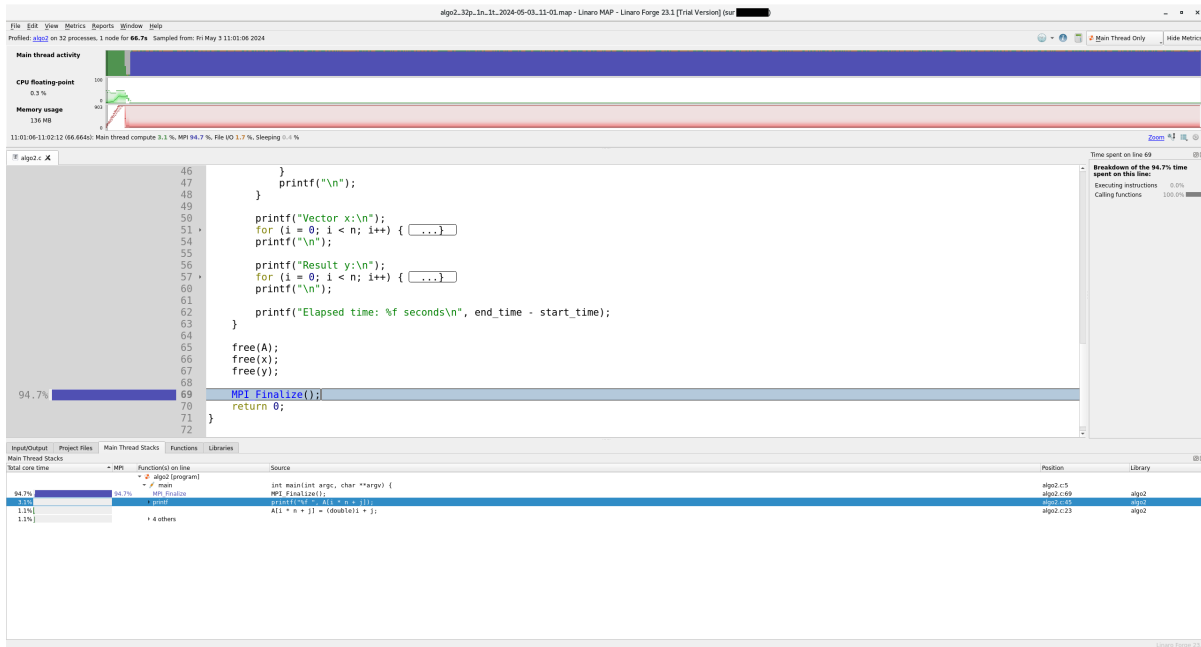
Fig. 1: Example of MAP profile

```bash
#!/bin/bash
#MSUB -r MyJob_Para                # Job name
#MSUB -q partition                 # Partition name
#MSUB -n 32                        # Number of tasks to use
#MSUB -T 1800                      # Elapsed time limit in seconds
#MSUB -o example_%I.o              # Standard output. %I is the job id
#MSUB -e example_%I.e              # Error output. %I is the job id
#MSUB -A <project>                 # Project ID
set -x
cd ${BRIDGE_MSUB_PWD}


module load linaro-forge
map --profile -n 32 ./a.out
```

Once the job has finished, a `.map` file should have been created. It can be opened from a remote desktop session or from an interactive session with the following command:

```
$ map <output_name>.map
```

Note: Linaro-forge MAP is a licenced product.

A full documentation is available in the installation path on the cluster. To open it:

```
$ evince ${MAP_ROOT}/doc/userguide-forge.pdf
```

Note: Allinea-forge has been renamed Arm-forge, which has then been renamed Linaro-forge.

20.5 Scalasca

Scalasca is a set of software which let you profile your parallel code by taking traces during the execution of the program. It is actually a wrapper that launches **Score-P** and **Cube**. This software is a kind of parallel **gprof** with more information. We present here an introduction of **Scalasca**. The generated output can then be opened with several analysis tools like **Periscope**, **Cube**, **Vampir**, or **Tau**.

Scalasca profiling requires 3 different steps:

- Instrumenting the code with **skin**
- Collecting profiling data with **scan**
- Examine collected information with **square**

20.5.1 Code instrumentation with Scalasca

First step for profiling a code with is instrumentation. You must compile your code by adding the wrapper before the call to the compiler. You need to load the **scalasca** module beforehand :

```
$ module load scalasca
$ skin mpicc -g -c prog.c
$ skin mpicc -o prog.exe prog.o
```

or for Fortran :

```
$ module load scalasca
$ skin mpif90 -g -c prog.f90
$ skin mpif90 -o prog.exe prog.o
```

You can compile for OpenMP programs:

```
$ skin ifort -openmp -g -c prog.f90
$ skin ifort -openmp -o prog.exe prog.o
```

You can profile hybrid MPI-OpenMP programs:

```
$ skin mpif90 -openmp -g -O3 -c prog.f90
$ skin mpif90 -openmp -g -O3 -o prog.exe prog.o
```

20.5.2 Simple profiling with Scalasca

Once the code has been instrumented with **Scalasca**, run it with **scan**. By default, a simple summary profile is generated.

Here is a simple example of a submission script:

```
#!/bin/bash
#MSUB -r MyJob_Para        # Job name
#MSUB -n 32                # Number of tasks to use
#MSUB -T 1800              # Elapsed time limit in seconds
#MSUB -o example_%I.o      # Standard output. %I is the job id
#MSUB -e example_%I.e      # Error output. %I is the job id
#MSUB -q <partition>       # Partition name
```

(continues on next page)

```
#MSUB -A <project>        # Project ID

set -x
cd ${BRIDGE_MSUB_PWD}

module load scalasca
export SCOREP_EXPERIMENT_DIRECTORY=scorep_profile.${BRIDGE_MSUB_JOBID}
scan ccc_mprun ./prog.exe
```

At the end of execution, the program generates a directory which contains the profiling files (the directory name is chosen with the SCOREP_EXPERIMENT_DIRECTORY environment variable):

```
$ tree scorep_profile.2871901
  |- profile.cubex
  `- scorep.cfg
```

The profile information can then be visualized with **square**:

```
$ module load scalasca
$ square scorep_profile.2871901
```
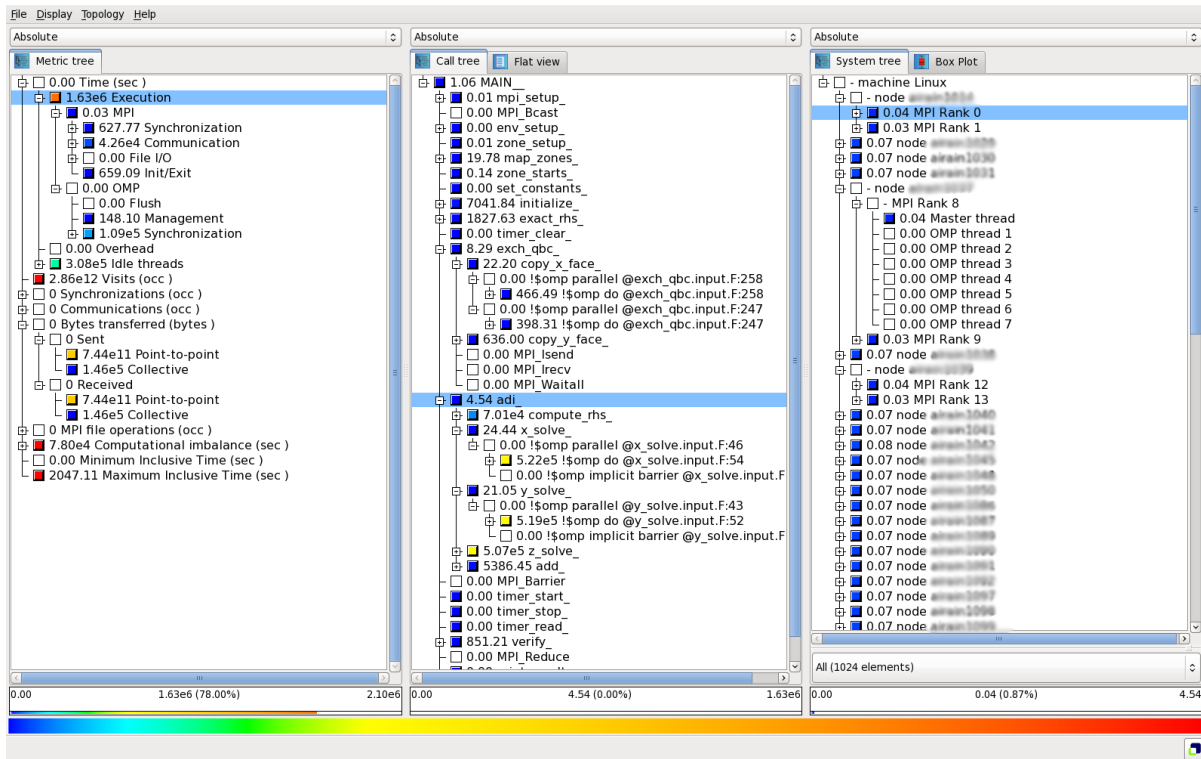

Fig. 2: Cube interface

20.5.3 Scalasca with PAPI

Score-P can retrieve the hardware counter with **PAPI**. For example, if you want retrieve the number of floating point operations:

```bash
#!/bin/bash
#MSUB -r MyJob_Para        # Job name
#MSUB -n 32                # Number of tasks to use
#MSUB -T 1800              # Elapsed time limit in seconds
#MSUB -o example_%I.o      # Standard output. %I is the job id
#MSUB -e example_%I.e      # Error output. %I is the job id
#MSUB -q <partition>       # Partition name
#MSUB -A <project>         # Project ID

set -x
cd ${BRIDGE_MSUB_PWD}

module load scalasca
export SCOREP_EXPERIMENT_DIRECTORY=scorep_profile.${BRIDGE_MSUB_JOBID}

export SCOREP_METRIC_PAPI=PAPI_FP_OPS
scan ccc_mprun ./prog.exe
```

Then the number of floating point operations will appear on the profile when you visualize it. The the syntax to use several papi counters is:

```bash
export SCOREP_METRIC_PAPI="PAPI_FP_OPS,PAPI_TOT_CYC"
```

20.5.4 Tracing application with Scalasca

To get a full trace there is no need to recompile the code. The same instrumentation is used for summary and trace profiling. To activate the trace collection, use the option -t of **scan**.

```bash
#!/bin/bash
#MSUB -r MyJob_Para        # Job name
#MSUB -n 32                # Number of tasks to use
#MSUB -T 1800              # Elapsed time limit in seconds
#MSUB -o example_%I.o      # Standard output. %I is the job id
#MSUB -e example_%I.e      # Error output. %I is the job id
#MSUB -q <partition>       # partition name
#MSUB -A <project>         # Project ID

set -x
cd ${BRIDGE_MSUB_PWD}

module load scalasca
export SCOREP_EXPERIMENT_DIRECTORY=scorep_profile.${BRIDGE_MSUB_JOBID}
scan -t ccc_mprun ./prog.exe
```

In that case, a file `traces.otf2` is created in the output directory with the summary. This profile trace can be opened with for example.

```
$ tree -L 1 scorep_profile.2727202
  |-- profile.cubex
  |-- scorep.cfg
  |-- traces/
  |-- traces.def
  `-- traces.otf2
```

> **Warning:** Generating a full trace may require a huge amount of memory.

Here is the best practice to follow:

- First start with a simple Scalasca analysis (without `-t`)

- Thanks to this summary, you can get an estimation of the size a full trace would take with the command:

```
$ square -s scorep_profile.2871901

Estimated aggregate size of event trace:                58GB
Estimated requirements for largest trace buffer (max_buf): 6GB
....
```

- If the *estimated aggregated size of event trace* seems excessive (it can easily reach several TB), you will need to apply filtering before recording the trace.

For more information on filtering and profiling options, check out the full documentation provided in the installation path:

```
$ module load scalasca
$ evince ${SCALASCA_ROOT}/share/doc/scalasca/manual/UserGuide.pdf
```

20.6 Vampir

`Vampir` is a visualization software that can be used to analyse **OTF traces**. The traces should have been generated before by one of the available profiling software such as `Score-P`.

20.6.1 Usage

To open a `Score-P` trace with `vampir`, just launch the graphical interface with the corresponding **OTF file**.

```
$ module load vampir
$ vampir scorep_profile.2871915/traces.otf2
```

It is not recommended to launch on the login nodes. An interactive session on compute nodes may be necessary. Also, the graphical interface may be slow. Using the Remote Desktop System service can help with that.

See the manual for full details and features of the vampir tool.
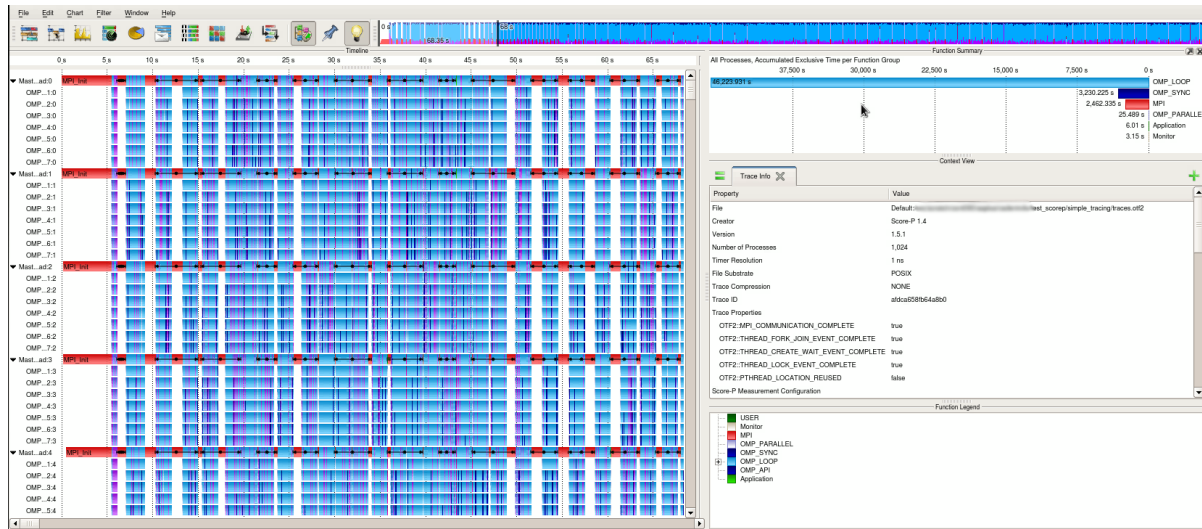
Fig. 3: Vampir window

20.6.2 Vampirserver

Traces generated by `Score-P` can be very large and can be very slow if you want to visualize these traces. `Vampir` provides `vampirserver`: it is a parallel program which uses CPU computing to accelerate Vampir visualization. Firstly, you have to submit a job which will launch on Irene nodes:

```
$ cat vampirserver.sh
#!/bin/bash
#MSUB -r vampirserver          # Job name
#MSUB -n 32                    # Number of tasks to use
#MSUB -T 1800                  # Elapsed time limit in seconds
#MSUB -o vampirserver_%I.o     # Standard output. %I is the job id
#MSUB -e vampirserver_%I.e     # Error output. %I is the job id
#MSUB -q partition             # Partition
#MSUB -A <project>             # Project ID


module load vampirserver
vampirserver start -n $((BRIDGE_MSUB_NPROC-1))
sleep 1700
```

```
$ ccc_msub vampirserver.sh
```

When the job is running, you will obtain this output:

```
$ ccc_mpp
USER ACCOUNT BATCHID NCPU QUEUE PRIORITY STATE RLIM  RUN/START SUSP OLD   NAME        ␣
→NODES
toto genXXX  234481  32   large 210332   RUN   30.0m 1.3m      -    1.3m vampirserver␣
→node1352


$ ccc_mpeek 234481
Found license file: /usr/local/vampir-7.5/bin/lic.dat
```

(continues on next page)

```
Running 31 analysis processes... (abort with Ctrl-C or vngd-shutdown)
Server listens on: node1352:30000
```

And a Vampir window should open.

Note: The `vampirserver` command runs in the background. So, without the call to **Vampir**, the job would be terminated immediately.

In our example, the Vampirserver master node is on node1352. The port to connect is 30000. Now you can use the graphical interface: start Vampir on a remote desktop service. Instead of clicking on "Open", you will click on "Remote Open":
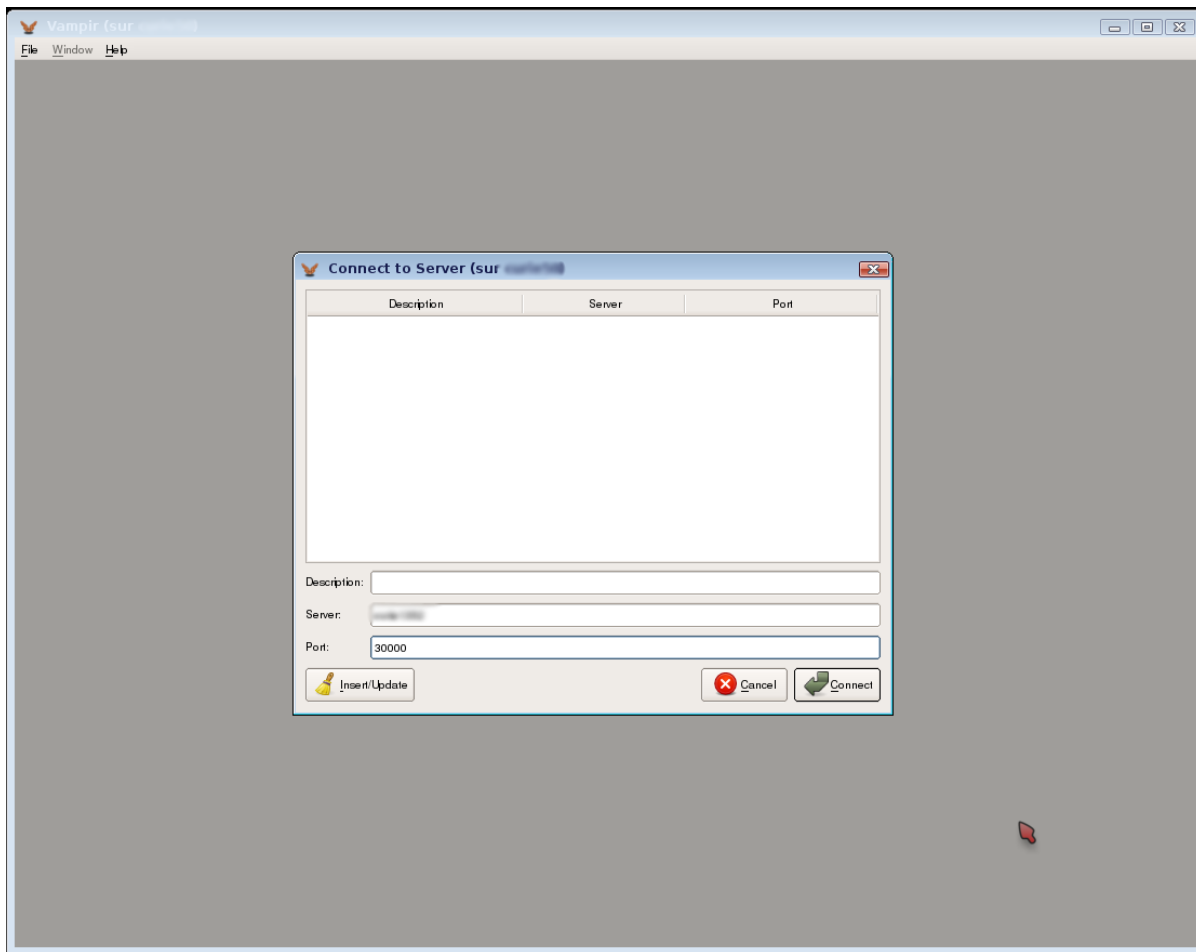
Fig. 4: Connecting to Vampirserver

Fill the server and the port, for instance *Server : node1352* and *Port : 30000* in the previous case. You will be connected to vampirserver. Then you can open an **OTF file** and visualize it.

Note:

- You can ask any number of processors you want: it will be faster if your profiling files are big. But be careful, it consumes your computing times.

• Don't forget to delete the Vampirserver job after your analyse.

20.7 Score-P

Score-P is a software system that provides measurement infrastructure for profiling, event trace recording, and analysis of HPC applications. Its goal is to simplify the analysis of the behavior of HPC software and to allow developers to find out where and why performance problems arise, where bottlenecks may be expected and where their codes offer room for further improvements. Score-P supports MPI, OpenMP, Pthreads, and CUDA, OpenCL and OpenACC and Fortran, C and C++ languages. It can take traces (OTF2) to be read by Vampir or Scalasca, and call-path profiles to be read by CUBE or TAU.

20.7.1 Instrumentation

Score-P is available through the **module** command:

```
$ module load scorep
```

You need to compile your code with scorep. Compile and link with:

- **scorep mpicc** for C code source files,
- **scorep mpicxx** for C++ code source files, and
- **scorep mpif90** for Fortran code source files,

20.7.2 Usage

The command to run scorep is:

```
$ ccc_mprun -n $NPROCS  ./my_app
```

When running the instrumented exectuable, the measurement system will create a directory called scorep-YYYYMMDD_HHMM_XXXXXXX, containing the date and time, and where XXXXXXXX is an additional identification number. The environment variables `SCOREP_ENABLE_TRACING` and `SCOREP_ENABLE_PROFILING` control whether event tracing or profiles are stored in this directory. By default, profiling is enabled and tracing is disabled.

After the completion of the execution, the requested data is available in the dedicated location, and can be visualized and analized by CUBE, TAU, or Vampir, for example.

For more information, see Score-P User Guide.

20.8 Darshan

Darshan is a scalable HPC I/O characterization tool. It is designed to profile I/O behavior with minimum overhead.

To run a program with Darshan profiling, there is no need to instrument or recompile the code.

- Load the **darshan module** : it tells **ccc_mprun** to wrap the I/O calls with the Darshan library.
- Specify where you want the darshan trace to be created by exporting the variable `DARSHAN_LOG_PATH`.
- The path must point to one directory of your home `$CCCHOME`.

Here is an example of a submission script to enable darshan profiling:

```bash
#!/bin/bash
#MSUB -r MyJob_Para        # Job name
#MSUB -n 32                # Number of tasks to use
#MSUB -T 1800              # Elapsed time limit in seconds
#MSUB -o example_%I.o      # Standard output. %I is the job id
#MSUB -e example_%I.e      # Error output. %I is the job id
#MSUB -q <partition>       # Partition
#MSUB -A <project>         # Project ID


set -x
cd ${BRIDGE_MSUB_PWD}

module load darshan
export DARSHAN_LOG_PATH=$CCCHOME


#The darshan module tells ccc_mprun to use the Darshan library.
ccc_mprun ./prog.exe
```

This will generate a trace in the specified directory. Here is the format of the output file: <USERNAME>_<BINARY_NAME>_<JOB_ID>_<DATE>_<UNIQUE_ID>_<TIMING>.darshan.gz.

Some scripts are available to post-process the output. For instance, **darshan-parser**, **darshan-job-summary.pl** and **darshan-summary-per-file.sh**.

- **darshan-job-summary.pl** will generate a graphical summary of the I/O activity for a job.

```
$ darshan-job-summary.pl *.darshan.gz
```

- **darshan-summary-per-file.sh** is similar except that it produces a separate pdf summary for every file accessed by the application. The summaries will be written in the directory specified as argument.

```
$ darshan-summary-per-file.sh *.darshan.gz output_dir
```

- **darshan-parser** gives a full, human readable dump of all information contained in a log file.

```
$ darshan-parser *.darshan.gz > example_output.txt
```

20.9 PAPI

PAPI is an API which allows you to retrieve hardware counters from the CPU. Here an example in Fortran to get the number of floating point operations of a matrix DAXPY:

```fortran
program main
 implicit none
 include 'f90papi.h'
 !
 integer, parameter :: size = 1000
 integer, parameter :: ntimes = 10
 double precision, dimension(size,size) :: A,B,C
 integer :: i,j,n
 ! Variable PAPI
```

(continues on next page)
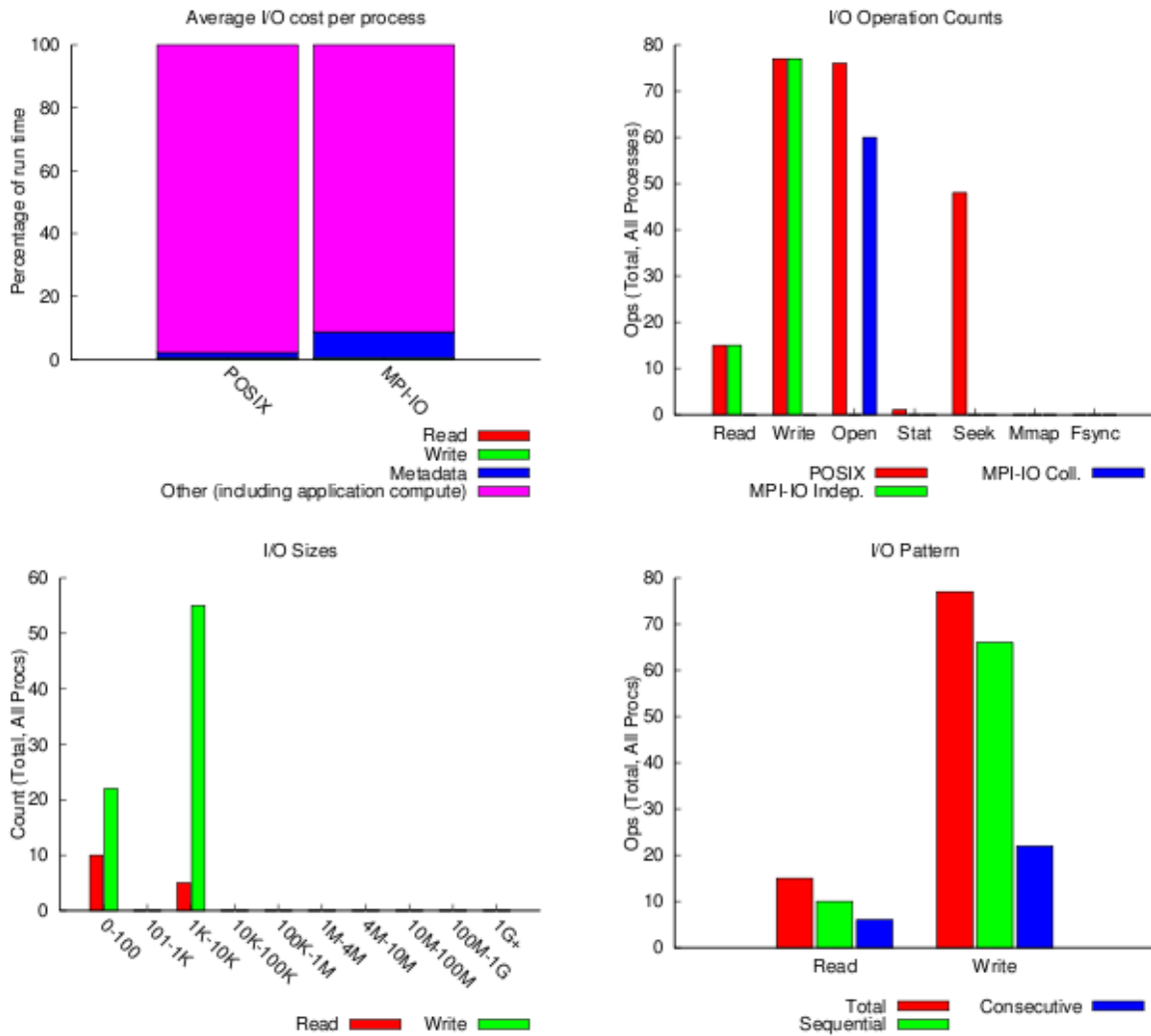
Fig. 5: Example of Darshan output

(continued from previous page)

```fortran
integer, parameter :: max_event = 1
integer, dimension(max_event) :: event
integer :: num_events, retval
integer(kind=8), dimension(max_event) :: values
! Init PAPI
call PAPIf_num_counters( num_events )
print *, 'Number of hardware counters supported: ', num_events
call PAPIf_query_event(PAPI_FP_INS, retval)
if (retval .NE. PAPI_OK) then
    event(1) = PAPI_TOT_INS
else
    ! Total floating point operations
    event(1) = PAPI_FP_INS
end if
! Init Matrix
do i=1,size
    do j=1,size
        C(i,j) = real(i+j,8)
        B(i,j) = -i+0.1*j
    end do
end do
! Set up counters
num_events = 1
call PAPIf_start_counters( event, num_events, retval)
! Clear the counter values
call PAPIf_read_counters(values, num_events,retval)
! DAXPY
do n=1,ntimes
    do i=1,size
        do j=1,size
            A(i,j) = 2.0*B(i,j) + C(i,j)
        end do
    end do
end do
! Stop the counters and put the results in the array values
call PAPIf_stop_counters(values,num_events,retval)
! Print results
if (event(1) .EQ. PAPI_TOT_INS) then
    print *, 'TOT Instructions:  ',values(1)
else
    print *, 'FP Instructions:  ',values(1)
end if
end program main
```

To compile, you have to load the PAPI module:

```
bash-4.00 $ module load papi/4.2.1
bash-4.00 $ ifort ${PAPI_CFLAGS} papi.f90 ${PAPI_LDFLAGS}
bash-4.00 $ ./a.out
 Number of hardware counters supported:          7
 FP Instructions:            10046163
```

To get the available hardware counters, you can type **papi_avail** command.

This library can retrieve the MFLOPS of a certain region of your code:

```fortran
program main
 implicit none
 include 'f90papi.h'
 !
 integer, parameter :: size = 1000
 integer, parameter :: ntimes = 100
 double precision, dimension(size,size) :: A,B,C
 integer :: i,j,n
 ! Variable PAPI
 integer :: retval
 real(kind=4) :: proc_time, mflops, real_time
 integer(kind=8) ::  flpins
 ! Init PAPI
 retval = PAPI_VER_CURRENT
 call PAPIf_library_init(retval)
 if ( retval.NE.PAPI_VER_CURRENT) then
     print*, 'PAPI_library_init', retval
 end if
 call PAPIf_query_event(PAPI_FP_INS, retval)
 ! Init Matrix
 do i=1,size
     do j=1,size
         C(i,j) = real(i+j,8)
         B(i,j) = -i+0.1*j
     end do
 end do
 ! Setup Counter
 call PAPIf_flips( real_time, proc_time, flpins, mflops, retval )
 ! DAXPY
 do n=1,ntimes
     do i=1,size
         do j=1,size
             A(i,j) = 2.0*B(i,j) + C(i,j)
         end do
     end do
 end do
 ! Collect the data into the Variables passed in
 call PAPIf_flips( real_time, proc_time, flpins, mflops, retval)
 ! Print results
 print *, 'Real_time: ', real_time
 print *, ' Proc_time: ', proc_time
 print *, ' Total flpins: ', flpins
 print *, ' MFLOPS: ', mflops
 !
end program main
```

and the output:

```
bash-4.00 $ module load papi/4.2.1
bash-4.00 $ ifort ${PAPI_CFLAGS} papi_flops.f90 ${PAPI_LDFLAGS}
bash-4.00 $ ./a.out
 Real_time:   6.1250001E-02
```

```
 Proc_time:    5.1447589E-02
 Total flpins:                100056592
 MFLOPS:     1944.826
```

If you want more precisions, you can contact us or visit PAPI website.

20.10 Memonit

Memonit is a memory profiling tool for HPC jobs. It traces memory usage and can be used to plot allocated memory for each process or node across the execution of an MPI program. To use the `memonit` tool:

Load the `memonit` module in your submission script:

```
module load memonit
```

If needed, change snapshot frequency:

```
export MEMONIT_DELAY=0:100 # snapshot every 100us, by default every 2s
```

Insert a call to the `memonit_collect` wrapper when launching your computation:

```
ccc_mprun memonit_collect ./my_job
```

Process the collected traces for post-treatment:

After your job is complete, records have been written to a collection directory. Use `memonit -a <prefix>` to consolidate and aggregate these records using the "prefix" name of the directory (if the directory name is `my_job.<JobId>.d`, use `my_job.<JobId>` as the prefix name), and `memonit -g` or `memonit_gui` to open the results with the graphic interface (GUI requires X forwarding). Both steps can be combined in a single command, such as `memonit -a -g <prefix>`.

Example of lines that you can insert into your submission script:

```
# Load dependencies
module load python

# Aggregate traces into my_job.db
memonit_aggregate -f my_job.metadata.json

# Analyse by mpi rank
memonit_gui -f my_job.db

# Or by node
memonit_gui -f my_job_bynode.db
```

20.11 Valgrind

Valgrind is an instrumentation framework for dynamic analysis tools. It comes with a set of tools for profiling and debugging codes.

To run a program with **valgrind**, there is no need to instrument, recompile, or otherwise modify the code.

Note: A tool in the Valgrind distribution can be invoked with the `--tool` option:

```
module load valgrind
valgrind --tool=<toolname> #default is memcheck
```

20.11.1 Callgrind

Callgrind is a profiling tool that records the call history among functions in a program's run as a call-graph. By default, the collected data consists of the number of instructions executed, their relationship to source lines, the caller/callee relationship between functions, and the numbers of such calls.

To start a profile run for a program, execute:

```
module load valgrind
valgrind --tool=callgrind [callgrind options] your-program [program options]
```

While the simulation is running, you can observe the execution with:

```
callgrind_control -b
```

This will print out the current backtrace. To annotate the backtrace with event counts, run:

```
callgrind_control -e -b
```

After program termination, a profile data file named `callgrind.out.<pid>` is generated, where pid is the process ID of the program being profiled. The data file contains information about the calls made in the program among the functions executed, together with Instruction Read (Ir) event counts.

To generate a function-by-function summary from the profile data file, use:

```
callgrind_annotate [options] callgrind.out.<pid>
```

20.11.2 Cachegrind

Cachegrind simulates how your program interacts with a machine's cache hierarchy and (optionally) branch predictor. To run **cachegrind** on a program *prog*, you must specify `--tool=cachegrind` on the **valgrind** command line:

```
module load valgrind
valgrind --tool=cachegrind prog
```

Branch prediction statistics are not collected by default. To do so, add the option `--branch-sim=yes`.

```
valgrind -tool=cachegrind --branch-sim=yes prog
```

One output file will be created for each process launched with **cachegrind**. To analyse the output, use the command **cg_annotate**:

```
$ cg_annotate <output_file>
```

cg_annotate can show the source codes annotated with the sampled values. Therefore, either use the option `--auto=yes` to apply to all the available source files or specify one file by passing it as an argument.

```
$ cg_annotate <output_file> sourcecode.c
```

For more information on Cachegrind, check out the official Cachegrind User Manual.

20.11.3 Massif

Massif measures how much heap memory a program uses. This includes both the useful space, and the extra bytes allocated for book-keeping and alignment purposes. Massif can optionally measure the stack memory.

As for the other Valgrind tools, the program should be compiled with debugging info (the `-g` option). To run **massif** on a program *prog*, the **valgrind** command line is:

```
module load valgrind
valgrind --tool=massif prog
```

The Massif option `--pages-as-heap=yes` allows to measure all the memory used by the program.

By default, the output file is called `massif.out.<pid>` (pid is the process ID), although this file name can be changed with the `--massif-out-file` option. To present the heap profiling information about the program in a readable way, run **ms_print**:

```
$ ms_print massif.out.<pid>
```

20.12 TAU (Tuning and Analysis Utilities)

TAU Performance System is a portable profiling and tracing toolkit for performance analysis of parallel programs written in Fortran, C, C++, UPC, Java, Python.

TAU (Tuning and Analysis Utilities) is capable of gathering performance information through instrumentation of functions, methods, basic blocks, and statements as well as event-based sampling. The instrumentation can be inserted in the source code using an automatic instrumentor tool based on the Program Database Toolkit (PDT), dynamically using DyninstAPI, at runtime in the Java Virtual Machine, or manually using the instrumentation API.

TAU's profile visualization tool, paraprof, provides graphical displays of all the performance analysis results, in aggregate and single node/context/thread forms. The user can quickly identify sources of performance bottlenecks in the application using the graphical interface. In addition, TAU can generate event traces that can be displayed with the Vampir, Paraver or JumpShot trace visualization tools.

20.12.1 Instrumentation

TAU is available through the **module** command:

```
$ module load tau
```

Specify programming model by setting `TAU_MAKEFILE` to one of `$TAU_MAKEFILEDIR/Makefile.tau-*`:

- `Makefile.tau-icpc-papi-mpi-cupti-pdt-openmp`

Compile and link with:

- **tau_cc.sh** for C code source files,
- **tau_cxx.sh** for C++ code source files, and
- **tau_f90.sh** for Fortran code source files,

20.12.2 Usage

The command to run TAU is:

```
$ ccc_mprun -n $NPROCS tau_exec ./a.out
```

Examine results with **paraprof/pprof**

```
$ pprof [directory_path]
```

or

```
$ paraprof
```
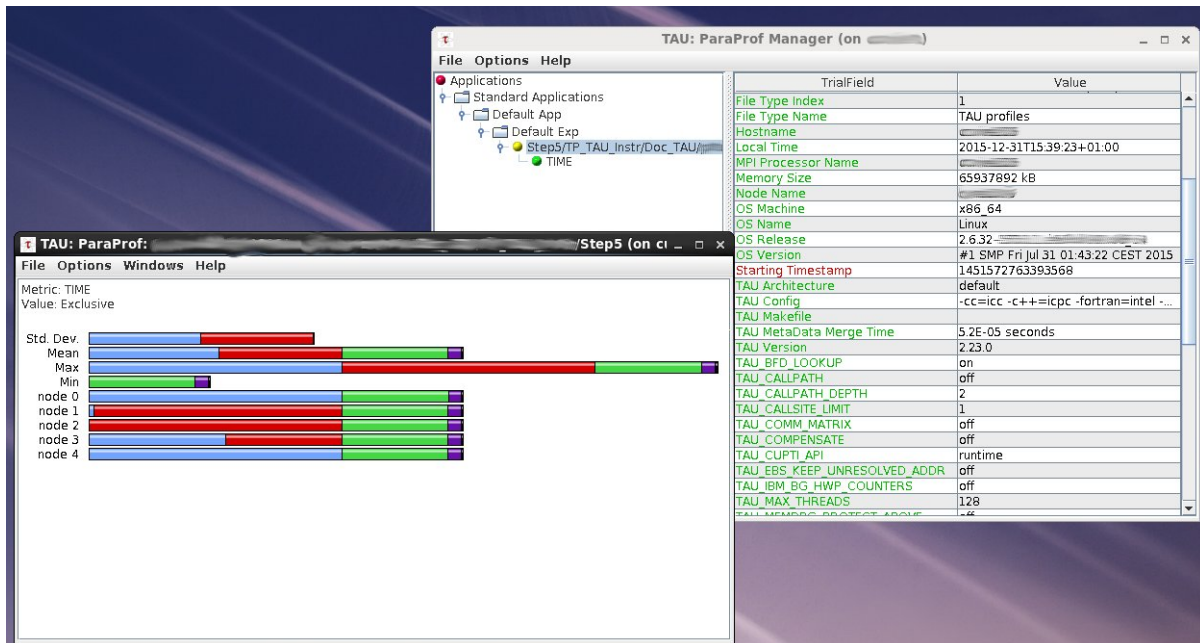

Fig. 6: Example of ParaProf window

Note: It's recommended to use the Remote Desktop session to use the graphical tools provided by tau (e.g. paraprof).

Environment variables control measurement mode `TAU_PROFILE`, `TAU_TRACE`, `TAU_CALLPATH` are available to tune the profiling settings.

For more information, see TAU User Guide.

20.13 Perf

Perf is a portable tool included in Linux kernel, it doesn't need to be loaded, doesn't need any driver and also works on all Linux platforms.

It is a performance analysis tool which displays performance counters such as the number of cache loads misses or branch loads misses.

- Perf cannot be fully launched on login nodes, you will need to execute it on compute node(s).

- In order to profile several processes with perf, you will need to differentiate the outputs by processe. Here is a simple wrapper to do it :

```
$ cat wrapper_perf.sh
#!/bin/bash
perf record -o out.${SLURM_PROCID}.data $@
```

- To run a command and gather performance counter statistics use **perf stat**. Here is a simple job example:

```
#!/bin/bash
#MSUB -n 4
#MSUB -c 12
#MSUB -T 400
#MSUB -q |Partition|
#MSUB -A <project>
#MSUB -x
#MSUB -E '--enable_perf'

export OMP_NUM_THREADS=${BRIDGE_MSUB_NCORE}

ccc_mprun wrapper_perf stat ./exe
```

- Informations will be stored in `out.<jobID>.data`:

```
$ cat out.0.data
# started on Wed Oct 11 14:01:14 2017


Performance counter stats for './exe':

  50342.076495      task-clock:u (msec)      #    1.000 CPUs utilized
  98,323            page-faults:u            #    0.002 M/sec
  155,516,791,925   cycles:u                 #    3.089 GHz
  197,715,764,466   instructions:u           #    1.27  insn per cycle


    50.348439743 seconds time elapsed
```

Now for more specifics counters:

- list of all performance counters with the command **perf list**

- Use the -e option to specify the wanted performance counters. For example, to get how much cache access misses within all cache access:

```
$ ccc_mprun perf stat -o perf.log -e cache-references,cache-misses ./exe
$ cat perf.log
# started on Wed Oct 11 14:02:52 2017


Performance counter stats for './exe':

  8,654,163,728      cache-references:u
  875,346,349        cache-misses:u                # 10.115 % of all cache refs


    52.710267128 seconds time elapsed
```

Perf can also record few counters from an executable and create a report:

- Use a job to create a report with

```
$ ccc_mprun -vvv perf record -o data.perf ./exe
...
[ perf record: Woken up 53 times to write data ]
[ perf record: Captured and wrote 13.149 MB data.perf (344150 samples) ]
...
```

- Read the report from the login node

```
$ perf report -i data.perf
Samples: 344K of event 'cycles:u', Event count (approx.): 245940413046
Overhead  Command  Shared Object       Symbol
 99.94%  exe      exe                 [.] mydgemm_
  0.02%  exe      [kernel.vmlinux]    [k] apic_timer_interrupt
  0.02%  exe      [kernel.vmlinux]    [k] page_fault
  0.00%  exe      exe                 [.] MAIN__
  ...
```

- You can also using call graphs with

```
$ ccc_mprun -vvv perf record --call-graph fp -o data.perf ./exe

$ perf report -i data.perf
Samples: 5K of event 'cycles:u', Event count (approx.): 2184801676
 Children     Self  Command  Shared Object           Symbol
+  66.72%    0.00%  exe      libc-2.17.so            [.] __libc_start_main
+  61.03%    0.03%  exe      libiomp5.so             [.] __kmpc_fork_call
-  60.96%    0.05%  exe      libiomp5.so             [.] __kmp_fork_call
    60.90% __kmp_fork_call
     - __kmp_invoke_microtask
          56.06% nextGen
          3.33% main
          1.35% __intel_avx_rep_memcpy
+  60.90%    0.03%  exe      libiomp5.so             [.] __kmp_invoke_microtask
```

(continues on next page)

```
+   56.06%   56.06% exe      exe                      [.] nextGen
+    8.98%    5.86% exe      exe                      [.] main
...
```

20.14 Extra-P

Extra-P is an automatic performance-modeling tool that can be used to analyse several SCOREP_EXPERIMENT_DIRECTORY generated with **Score-P**. The primary goal of this tool is identify scalability bugs but due to his multiple graphics outputs, it's also useful to make reports.

20.14.1 Usage

To analyse the scalability of an algorithm you need to generate several SCOREP_EXPERIMENT_DIRECTORY, for example you can launch this submission script:

```
#!/bin/bash
#MSUB -r npb_btmz_scorep
#MSUB -o npb_btmz_scorep_%I.o
#MSUB -e npb_btmz_scorep_%I.e
#MSUB -Q test
#MSUB -T 1800    # max walltime in seconds
#MSUB -q haswell
#MSUB -A <project>

cd $BRIDGE_MSUB_PWD

# benchmark configuration
export OMP_NUM_THREADS=$BRIDGE_MSUB_NCORE

# Score-P configuration
export SCOREP_EXPERIMENT_DIRECTORY=scorep_profile.p$p.r$r
PROCS=$BRIDGE_MSUB_NPROC
EXE=./exe

ccc_mprun -n $PROCS $EXE
```

from a bash script (4 runs on each scripts with 8, 16, 32 and 64 cores):
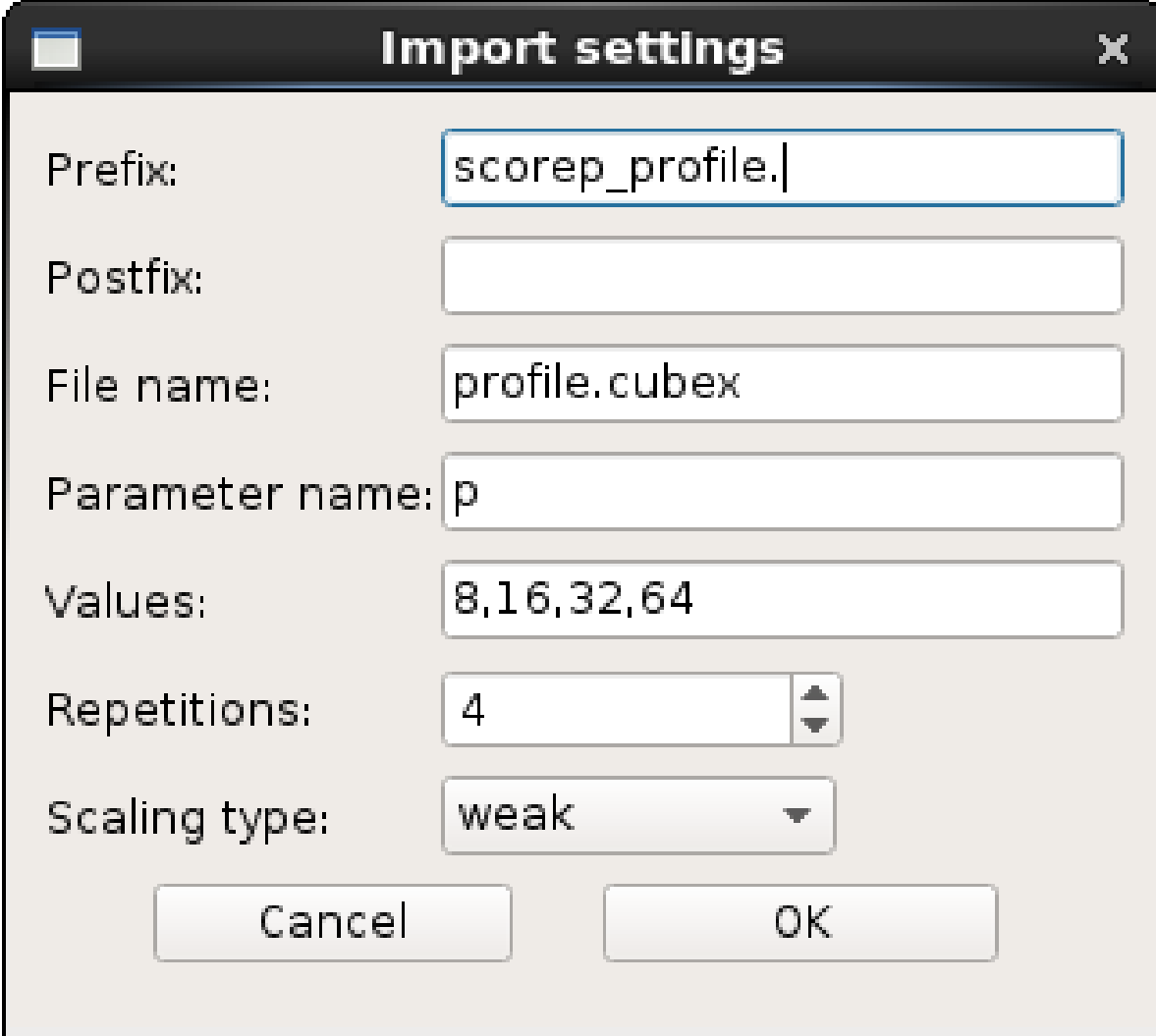
```
p=4 n=2 c=2 r=1 ccc_msub -n 2 -c 2 submit_global.msub
p=4 n=2 c=2 r=2 ccc_msub -n 2 -c 2 submit_global.msub
p=4 n=2 c=2 r=3 ccc_msub -n 2 -c 2 submit_global.msub
p=4 n=2 c=2 r=4 ccc_msub -n 2 -c 2 submit_global.msub
p=8 n=2 c=4 r=1 ccc_msub -n 2 -c 4 submit_global.msub
[...]
p=32 n=8 c=4 r=4 ccc_msub -n 8 -c 4 submit_global.msub
p=64 n=8 c=8 r=1 ccc_msub -n 8 -c 8 submit_global.msub
p=64 n=8 c=8 r=2 ccc_msub -n 8 -c 8 submit_global.msub
p=64 n=8 c=8 r=3 ccc_msub -n 8 -c 8 submit_global.msub
p=64 n=8 c=8 r=4 ccc_msub -n 8 -c 8 submit_global.msub
```

Once these folders are generated load **Extra-P**:

```
$ module load extrap
```

launch it and open the parent folder. The software detects automatically your files:

On Metric section choose "time" and you will get the time of each commands. You can also right clic on the graph and choose "Show all data points", you will see the time of all your runs:
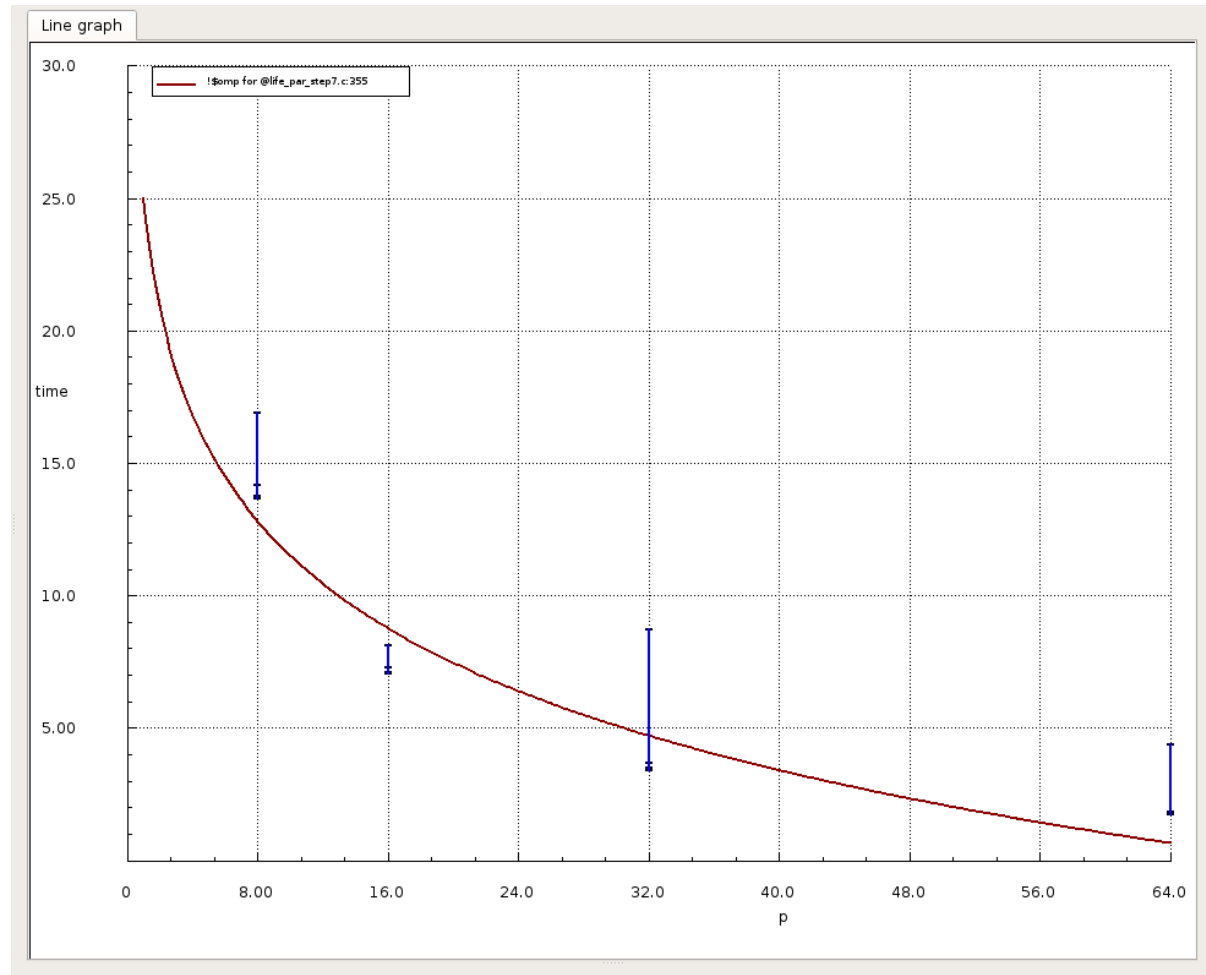
Note: These graphics only give the time of your application, not any speedup.

Extra-P also list all MPI and OpenMP requests, you can select what you want and check scalability:

Sev	Callpath	Cor	Value	RSS	Adj. R²	SMAPE
	main	1	9.842×10^{-03}-1.059×1...	3.528×10^{-07}	-0.110	2.210
	MPI_Init	1	0.141+1.389×10^{-03}*(...	2.245×10^{-03}	0.978	7.989
	MPI_Comm_rank		3.940×10^{-07}+$5.830 \times$...	2.686×10^{-16}	0.374	1.703
	MPI_Comm_size	1	9.09595×10^{-08}	5.509×10^{-15}	0.000	33.035
	MPI_File_open	1	3.818×10^{-03}+$1.284 \times$...	1.293×10^{-06}	0.998	10.125
	MPI_File_read_at	1	-8.015×10^{-03}+$3.311 \times$...	1.071×10^{-05}	0.755	27.052
	MPI_File_seek	1	2.584×10^{-06}+$4.546 \times$...	2.908×10^{-14}	0.160	2.612
	MPI_File_read	1	-2.437×10^{-03}+$1.771 \times$...	2.994×10^{-06}	0.759	12.087
	MPI_File_close	1	-1.981×10^{-03}+$2.227 \times$...	1.210×10^{-07}	0.991	40.477
	▶ writeState	1	3.820×10^{-05}-1.534×1...	2.004×10^{-12}	0.782	2.097
	MPI_Isend	1	1.496×10^{-03}+$8.019 \times$...	7.479×10^{-07}	0.717	8.842
	MPI_Irecv	1	1.136×10^{-03}+$6.165 \times$...	3.736×10^{-07}	0.754	7.971
	MPI_Wait	1	-0.539+0.214*$\log_2(p)$	0.346	0.097	56.234
	▼ !$omp parallel...		4.119×10^{-03}-6.691×1...	4.821×10^{-09}	0.867	0.692
	▶ !$omp for ...	1	24.999-4.059*$\log_2(p)$	6.446	0.891	37.029
	▶ !$omp for ...	1	0.0613047	8.717×10^{-03}	0.000	68.455
	!$omp impl...		5.037×10^{-03}-8.397×1...	1.396×10^{-09}	0.974	0.286
	▶ checkImage	1	0.310-0.049*$\log_2(p)$	5.075×10^{-04}	0.939	21.978
	MPI_Finalize	1	7.651×10^{-05}+$8.151 \times$...	2.950×10^{-10}	0.999	8.544

At least, you can choose to use strong scaling and see the efficiency of an algorithm:

```
Import settings                              ✕

Prefix:             scorep_profile.

Postfix:

File name:          profile.cubex

Parameter name:  p

Values:             8,16,32,64

Repetitions:        4                        ▲
                                             ▼

Scaling type:       strong                 ▼

        Cancel                    OK
```

20.15 AMD µProf

AMD µprof ("MICRO-prof") is a performance analysis tool-suite for x86 based applications running on Linux. It provides performance metrics for AMD "Zen"-based processors and AMD Instinct MI Series accelerators. AMD µProf enables the developer to better understand the performance bottlenecks, optimization scope, and evaluate improvements. It offers the following functionalities:

- Performance Analysis (CPU Profile): To identify runtime performance bottlenecks of the application.
- System Analysis: To monitor system performance metrics, such as IPC and memory bandwidth.
- Live Power Profile: To monitor thermal and power characteristics of the system.

```
$ module load uprof/3.4
```

Be careful, AMD µProf works in the **/tmp** directory. Therefore, you should copy your input files into it and copy your output files back to your base folder.

For CLI usage, here below is an example of submission script with several features of µprof you can test:

Example of submission script:

```
$ #!/bin/bash
#MSUB -r MyJob_Para # Job name
#MSUB -q partition # Partition name
#MSUB -n 32 # Number of tasks to use
#MSUB -T 1800 # Elapsed time limit in seconds
#MSUB -o example_%I.o # Standard output. %I is the job id
#MSUB -e example_%I.e # Error output. %I is the job id
#MSUB -A <project> # Project ID
set -x
cd ${BRIDGE_MSUB_PWD}
module load uprof

# Output directory
mkdir uprof_${BRIDGE_MSUB_JOBID}

# Copy all your executables in /tmp because AMDuProf works on it
cp -R ${BRIDGE_MSUB_PWD}/Examples/AMDTClassicMatMul /tmp/

# Profiling Time-Based Sampling
ccc_mprun AMDuProfCLI collect --config tbp -o /tmp/cpuprof-tbp -a Examples/
↪AMDTClassicMatMul/bin/AMDTClassicMatMul-bin
wait

# Generate Reports
ccc_mprun AMDuProfCLI report -i /tmp/cpuprof-tbp.caperf -o /tmp/tbp_report.csv
wait

# Before being deleted, copy all your output CSV files from /tmp into your base directory
cp -R /tmp/tbp_report.csv/cpuprof-tbp ${BRIDGE_MSUB_PWD}/uprof_${BRIDGE_MSUB_JOBID}/
```

For interactive profiling, you can use **AMDuProf** in your submission script instead of **AMDuProfCLI**.

```
$ ccc_mprun AMDuProf Examples/AMDTClassicMatMul/bin/AMDTClassicMatMul-bin
```

First, connect to the interactive **AMDuProf** session. Then, select the instance you want to profile and your application target.

After a short time of profiling, you will see a detailed analysis of all of your execution instances.

AMDCpuTopology: Command-line tool to get the CPU topology of AMD Processors.

```
$ AMDCpuTopology
----------------------------------------------
Package Numa CCD CCX Core Thread
----------------------------------------------
0 0 0 0 0 0
0 0 0 0 0 64
... ... ... ... ... ....
```

Note: Please run all the following AMD μProf commands with **ccc_mprun** in your submission script to avoid root errors.

Profile Duration : 2s

CPU_TIME

Hot Functions
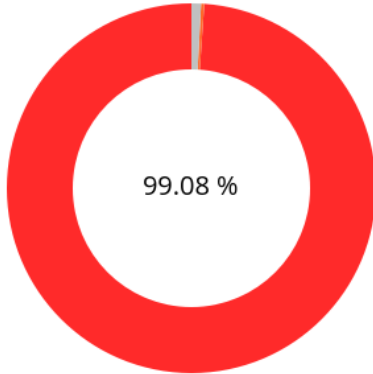
Function	CPU_TIME [seconds]	Module
classic_multiply_matrices()	1.405	AMDTClassicMatMul-bin
initialize_matrices()	0.004	AMDTClassicMatMul-bin
_raw_spin_unlock_irqrestore	0.002	[vmlinux]
clear_page_erms	0.001	[vmlinux]
file_ra_state_init	0.001	[vmlinux]
Others	0.005	N/A

Fig. 7: Example of AMD µProf

Fig. 8: Some features of AMD µProf

AMDuProfCLI: Command-line tool for µprof with profiling on AMD CPUs and GPUs.

- **AMDuProfCLI collect --config ...**: Run a command and collect the performance profile data in .caperf file.

- **AMDuProfCLI timechart --event ... --interval ... --duration ...**: Tool to visualize total system behavior during a workload like power, thermal and frequency.

- **AMDuProfCLI report -s event=... -i mon_programme.caperf**: Process .caperf profile-data file created by **AMDuProfCLI collect** and generates the profile report.

- **AMDuProfCLI info ...**: Run a command and display generic information about system, CPU etc.

Run **AMDuProfCLI <COMMAND> -h** for more information on a specific command.

AMDuProf: Graphical User Interface for intuitive visualization of performance metrics.

- Timelines: Hardware-accelerated views; thread callstack tracing.

- GPU Metrics: SMI data (power, temp, VRAM); HIP/HSA summary tables.

- Source View: Reorder by execution/line number for optimized code analysis.

20.16 Gprof

Gprof produces an execution profile of C/C++ and Fortran programs. The effect of called routines is incorporated in the profile of each caller. The profile data is taken from the call graph profile file (gmon.out default) which is created by programs that are compiled with the -pg option. Gprof reads the given object file (the default is "a.out") and establishes the relation between its symbol table and the call graph profile from gmon.out.

- To load gprof:

```
$ module load gprof
```

- To compile with -pg option

```
$ icc -pg hello.c
```

- If your application is MPI, set environment variable to rename gmon files and enable one file per process

```
$ export GMON_OUT_PREFIX='gmon.out-'`/bin/uname -n`
```

Files generated will be named gmon.out-<hostname>.<pid> (ie: gmon.out-node1192.56893).

- To generate call graph profile gmon.out

```
$ ./a.out
```

- To display flat profile and call graph

```
$ gprof a.out gmon.out
```

- To display only the flat profile

```
$ gprof -p -b ./a.out gmon.out

Flat profile:
Each sample counts as 0.01 seconds.
```

(continues on next page)

```
 %    cumulative    self               self     total
time     seconds   seconds    calls    s/call   s/call  name
67.72      10.24     10.24        1     10.24    10.24  bar
33.06      15.24      5.00                             main
 0.00      15.24      0.00        1      0.00    10.24  foo
```

- To display the flat profile of one specific routine

```
$ gprof -p<routine> -b ./a.out gmon.out
```

- To display only the call graph

```
$ gprof -q -b ./a.out gmon.out

Call graph (explanation follows)
index % time    self  children    called     name
                                                <spontaneous>
[1]    100.0    5.00   10.24                 main [1]
                0.00   10.24      1/1             foo [3]
-----------------------------------------------
               10.24    0.00      1/1             foo [3]
[2]     67.2   10.24    0.00      1         bar [2]
-----------------------------------------------
                0.00   10.24      1/1             main [1]
[3]     67.2    0.00   10.24      1         foo [3]
               10.24    0.00      1/1             bar [2]
-----------------------------------------------
```

- To display the call graph of one specific routine

```
$ gprof -q<routine> -b ./a.out gmon.out
```

- To generate a graph from data, see *gprof2dot and gprof*

20.17 Gprof2dot

Gprof2dot is an utility which converts profile data into a dot graph. It is compatible with many profilers.
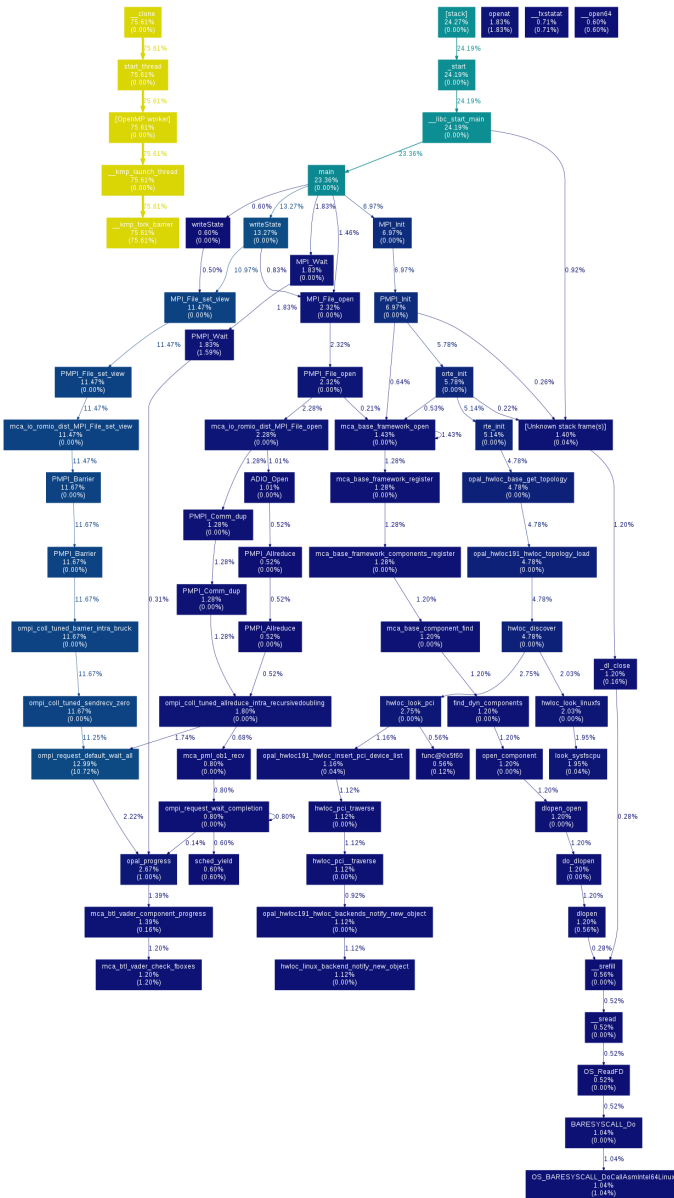
20.17.1 Use with gprof

- Load modules gprof AND gprof2dot

```
$ module load gprof gprof2dot
```

- Be sure to compile your application with `-pg` option

```
$ icc -pg hello.c
```

- Generate the call graph profile `gmon.out` (Run the application one time)

```
$ ./a.out
```

- Generate the dot graph in a PNG image

```
$ gprof ./a.out gmon.out | gprof2dot.py | dot -Tpng -o a.png
```

20.17.2 Use with VTune in command line

- Load modules VTune AND gprof2dot

```
$ module load vtune gprof2dot
```

- Use VTune to collect data

```
$ ccc_mprun amplxe-cl -collect hotspots -result-dir output -- ./exe
```

- Transform data in a gprof-like file

```
$ amplxe-cl -report gprof-cc -result-dir output -format text -report-output output.txt
```

- Generate the dot graph

```
$ gprof2dot.py -f axe output.txt | dot -Tpng -o output.png
```

For more information about gprof2dot, see GitHub gprof2dot.

20.18 cProfile: Python profiler

cProfile is a profiler included with Python. It returns a set of statistics that describes how often and for how long various parts of the program executed.

First, load *python3* module:

```
$ module load python3
```

Then, use *-m cProfile* option of the *python3* command:

```
$ python3 -m cProfile monte_carlo_pi.py
```

```
Sorting by cumulative:
Estimated value of pi: 3.1413716
        40001353 function calls (40001326 primitive calls) in 17.496 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      3/1    0.000    0.000   17.496   17.496 {built-in method builtins.exec}
        1    0.000    0.000   17.496   17.496 monte_carlo_pi.py:1(<module>)
        1    0.000    0.000   17.489   17.489 monte_carlo_pi.py:16(main)
        1    9.232    9.232   17.489   17.489 monte_carlo_pi.py:3(estimate_pi)
 20000000    6.687    0.000    8.257    0.000 random.py:415(uniform)
```

The output can be ordered by several keys, use the *-s* option to select:

```
$ python3 -m cProfile -s cumulative monte_carlo_pi.py
```

Here is a table of acceptable keys:

Sort Key	Description
calls	Number of times the function was called
cumulative	Total time spent in the function and all the functions it calls
cumtime	Total time spent in the function and all the functions it calls
file	Filename where the function is defined
filename	Filename where the function is defined
module	Filename where the function is defined
ncalls	Number of times the function was called
pcalls	Number of primitive (i.e., not induced via recursion) function calls
line	Line number in the file where the function is defined
name	Name of the function
nfl	Combination of function name, filename, and line number
stdname	Standard function name (a more readable version of 'nfl')
time	Internal time spent in the function excluding time made in calls to sub-functions
tottime	Internal time spent in the function excluding time made in calls to sub-functions

20.18.1 Functions profiling

cProfile is a powerful tool for profiling Python scripts. However, it's not always efficient to profile an entire script, especially if you have a specific function suspected to be a performance bottleneck.

In computational programs, a heavy-lifting function, such as one performing complex calculations or data manipulations, often consumes a significant portion of the execution time. Profiling this specific function can provide more granular insights, assisting in identifying areas for optimization.

Here's how to use cProfile for a specific function:

```python
import cProfile

def my_compute_func():
    # Complex computation function

cProfile.run('my_compute_func()', sort='cumulative')
```

Targeted profiling helps optimize the most time-consuming parts of your code, potentially resulting in substantial improvements in the overall execution time.

20.18.2 cProfile and mpi4py

It is possible to profile MPI code using cProfile, but it can be a bit more complex than profiling a single process Python script.

When an MPI program is run, multiple processes are created, each with its own Python interpreter. If you just run cProfile as usual, each process will attempt to write its profiling data to the same file, which will lead to problems.

To avoid this, you can modify your MPI script to write the profiling data from each process to a different file. Here is an example using mpi4py:

```python
from mpi4py import MPI
import cProfile
```

```python
def main(rank):
    # Your MPI code here
    pass


if __name__ == "__main__":
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()
    cProfile.run('main(rank)', f'output_{rank}.pstats')
```

Please remember that this way, the result is a number of different files each representing the profiling results of one MPI process. They won't give you a total view of the time spent across all processes, but you will be able to see what each individual process spent its time on.

20.18.3 cProfile and gprof2dot

cProfile output can be formatted into reports via the pstats module:

```
$ python3 -m cProfile -o output.pstats monte_carlo_pi.py
```

The pstats file is readable by *gprof2dot*:

```
$ module load gprof2dot
$ gprof2dot.py -f pstats output.pstats -o output.dot
$ dot -Tpng output.dot -o output.png
```

Then, display *output.png* with *eog* or *display* command.

20.19 Nsight System

nsys is a profiler and graphical profile data analysis tool for CUDA applications. It is installed with the Nvidia HPC Software Development Kit and can be accessed through the `nvhpc` module.

In order to profile your application, you can use the following startup command in a job script :

```
ccc_mprun nsys profile --trace=openacc,cuda,nvtx --stats=true ./set_visible_device.sh ./
→myprogram`
```

> **Warning:** A known bug affects nsys MPI auto-detection feature when using OpenMPI 4.1.x versions which produces a segmentation fault. In order to circumvent it, use the `--mpi-impl=openmpi` option when using `--trace=mpi`.

During execution, nsys will creates *report<n>.sqlite* and *report<n>.nsys-rep* files.

You may then use the command line or graphical interface to analyse and display these results. **nsys-ui** starts up the graphical interface. Use `nsys-ui path/to/report*.nsys-rep` to open up the nsys report files and navigate the collected traces.
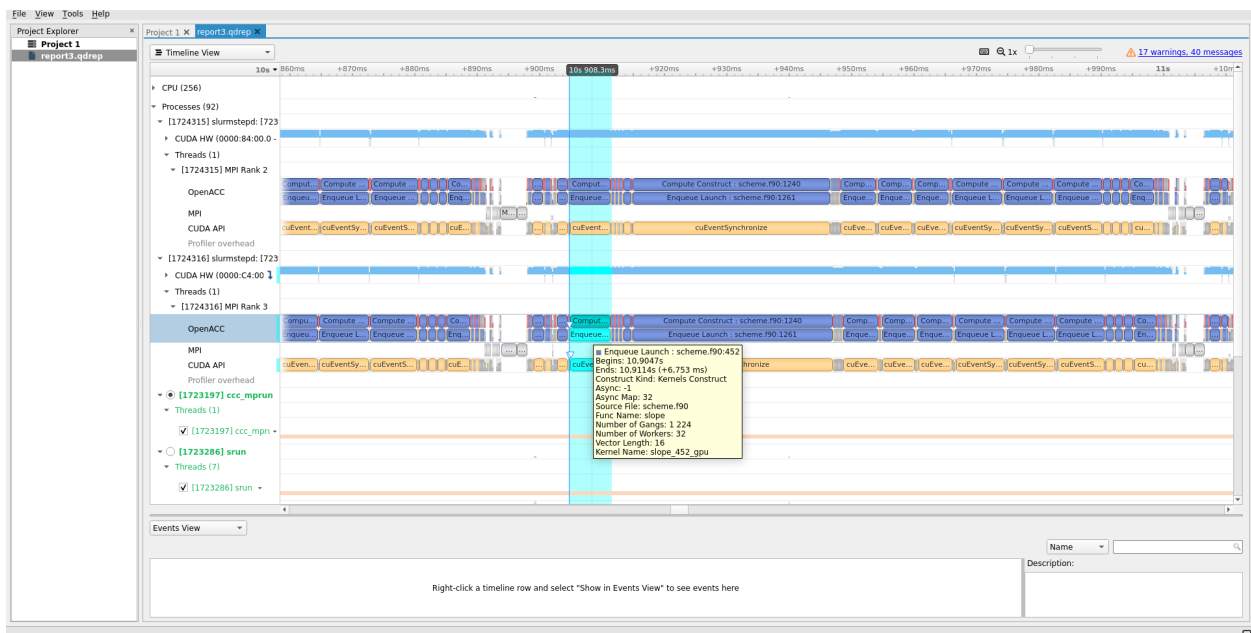
For convenience, you may install **Nsight Systems** on your own computer and copy the instrumentation data there for displaying. **Nsight Systems** is freely available to members of the *NVIDIA Developer Program* which is a free registration. Note that you will need to select the right version for download. Use `nsys --version` to know which one is being used in your profiling jobs.

The **nsys-ui** graphical interface can also be used to launch computations directly on nodes reserved, eg. using `ccc_mprun -K`.

Refer to the online Nsight Systems documentation or to the `nsys --help` command for more information

> **Warning:** When using *nsys*, the */tmp* directory might become saturated, leading to process failures. To avoid this, it is recommended to redirect temporary files to another location by modifying the *TMPDIR* environment variable.

For example:
```
export TMPDIR=$CCCSCRATCHDIR
```

20.20 Nsight Compute

ncu is a cuda kernel profiler for analysing GPU occupancy, and the usage of memory and compute units. It features advanced performance analysis tools and a graphical reporting interface providing useful hints and graphs.

In order to profile your application kernels, you may use the following startup command in a job script :

```
ccc_mprun ncu --call-stack --nvtx --set full -s 20 -c 20 --target-processes all -o
→report_%q{SLURM_PROCID} ./set_visible_device.sh ./myprogram
```

This example will generate one report file, named *report_X.ncu-rep*, for each process started by the application.

It uses the `full` argument to the `--set` option, which triggers the collection of a set of metrics.

You may use other values for the `--set` option for specifying a chosen set of sections, use `--section` for enabling individual report sections and the associated metrics, or use `--metric` for activating the collection of individual metrics. Use `ncu --list-sets`, `ncu --list-sections` and `ncu --list-metrics` respectively for more informations.

The option `-s 20` ignores the first 20 kernels executed, in an attempt to avoid gathering information on kernels which are not representative of the general workload, such as those in charge of data initialization.

The option `-c 20` enables the profiling and collection of selected performance metrics for the next 20 kernels, providing control over the general overhead of profiling the executed program.

For more information, run `ncu --help` or refer to the online Nsight Compute CLI documentation.

Use the `ncu-ui` command to open the generated reports in the Nsight Compute GUI, and access the various performance analysis, hints, and graphs.

ID	Function Name	Demangled Name	Process	Device Name	Grid Size	Block Size	Cycles [cycle]	Duration [usecond]	Compute Throughput [%]	Memory Throughput [%]
0	checkImage_34_gpu	checkImage_34_gpu	[786099] life_par_st...	NVIDIA A100-SXM4-80GB	65535, 1, 1	128, 1, 1	291,667	233.38	40.58	85.85
2	main_377_gpu	main_377_gpu	[786099] life_par_st...	NVIDIA A100-SXM4-80GB	65535, 1, 1	128, 1, 1	903,036	710.05	73.32	61.42
3	main_383_gpu	main_383_gpu	[786099] life_par_st...	NVIDIA A100-SXM4-80GB	65535, 1, 1	128, 1, 1	597,303	470.46	17.27	82.08
4	checkImage_34_gpu	checkImage_34_gpu	[786099] life_par_st...	NVIDIA A100-SXM4-80GB	65535, 1, 1	128, 1, 1	291,781	232.83	40.57	85.80
6	main_377_gpu	main_377_gpu	[786099] life_par_st...	NVIDIA A100-SXM4-80GB	65535, 1, 1	128, 1, 1	903,026	710.43	73.32	61.41
7	main_383_gpu	main_383_gpu	[786099] life_par_st...	NVIDIA A100-SXM4-80GB	65535, 1, 1	128, 1, 1	595,202	469.41	17.33	82.35
8	checkImage_34_gpu	checkImage_34_gpu	[786099] life_par_st...	NVIDIA A100-SXM4-80GB	65535, 1, 1	128, 1, 1	291,160	233.44	40.66	86.00

For more information, please refer to the online Nsight Compute documentation

POST-PROCESSING

21.1 Gnuplot

The Gnuplot homepage provides links to all the documentation most users will need.

You can use **gnuplot** from a Remote Desktop System session or from a compute node (please avoid login nodes)

To start **gnuplot** from a Remote Desktop System session:

- Start your Remote Desktop System session

- Open a terminal

```
module load gnuplot
gnuplot
```

To start **gnuplot** from a compute node:

- Log into the login node, with X redirection (**ssh -X login@login-node**).

- Load the appropriate module:

```
module load gnuplot
```

- Start an interactive session from a compute node with X11 export and launch **gnuplot**:

```
ccc_mprun -Xfirst -T3600 -p <partition> -A <project> -s
gnuplot
```

- An example of **gnuplot** prompt:

```
$ gnuplot

    G N U P L O T
    Version x.y patchlevel z    last modified 2020-12-01

    Copyright (C) 1986-1993, 1998, 2004, 2007-2020
    Thomas Williams, Colin Kelley and many others

    gnuplot home:     http://www.gnuplot.info
    faq, bugs, etc:   type "help FAQ"
    immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'unknown'
gnuplot>
```
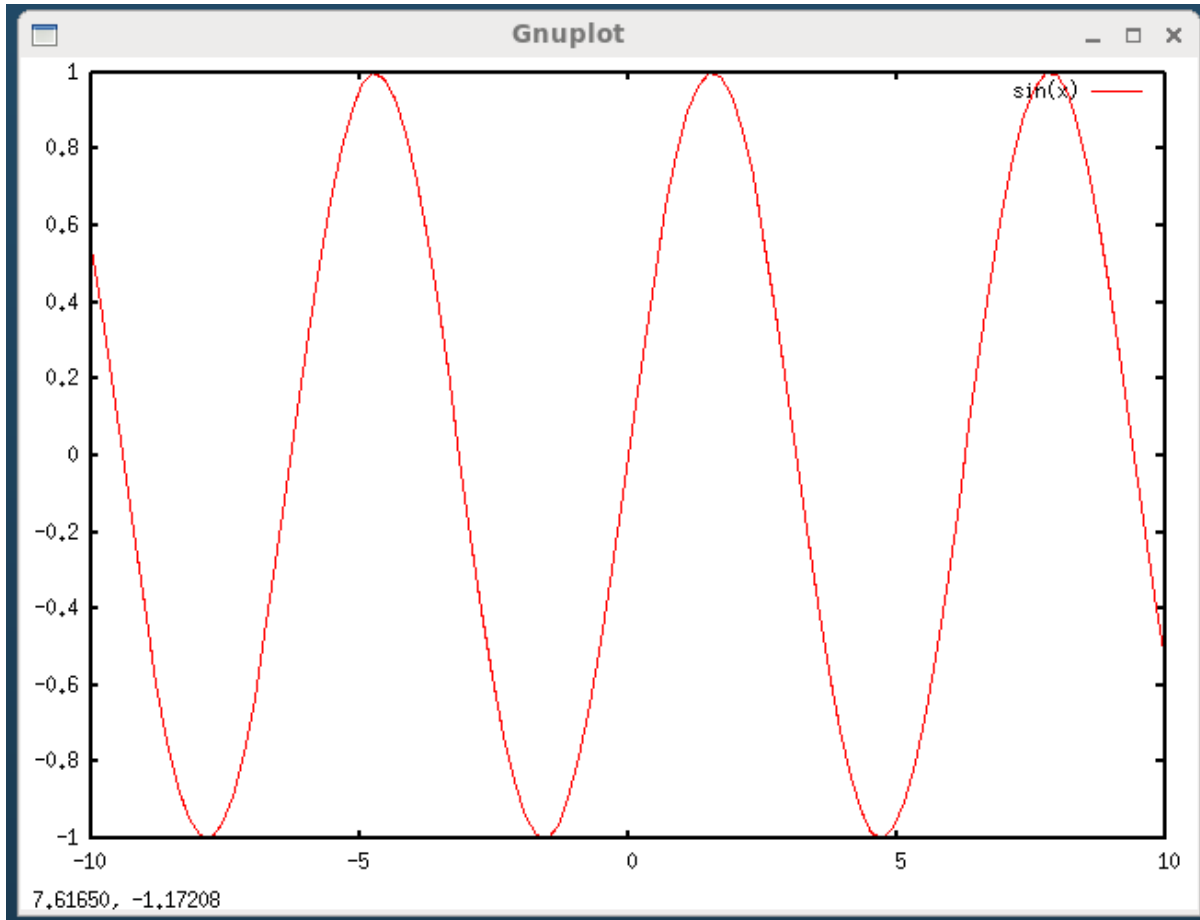
Fig. 1: Example of Gnuplot output (display)

- If your **gnuplot** session is ressource intensive, you may add the `-x` option to ensure the node is allocated to you only.

21.2 Xmgrace

You can use **xmgrace** from a Remote Desktop System session or from a compute node (please avoid login nodes)

To start **xmgrace** from a Remote Desktop System session:

- Start your Remote Desktop System session

- Open a terminal

```
module load grace
xmgrace
```

To start **xmgrace** from a compute node:

- Log into the login node, with X redirection and thrusted X11 forwarding

```
ssh -Y login@login-node
```

- Find **xmgrace**:

```
module load grace
```

- Start **xmgrace** from an exclusive compute node:

```
ccc_mprun -Xfirst -x -Q normal -p <partition> xmgrace -A <project>
```

The program **xmgrace** calls firefox to display its help contents that are available as HTML. Since all nodes don't have browsers, you may see:

- Tutorial and manuals may be found on website of grace

21.3 Tecplot

You can use Tecplot from a Remote Desktop System session or from a compute node (please avoid login nodes)

To start tecplot from a Remote Desktop System session:

- Start your Remote Desktop System session

- Open a terminal

```
module load tecplot
tec360
```

To start Tecplot from a compute node:

- Log into the login node, with X redirection (**ssh -X login@login-node**).

- Find tecplot:

```
module load tecplot
```

- Start Tecplot from an exclusive compute node:
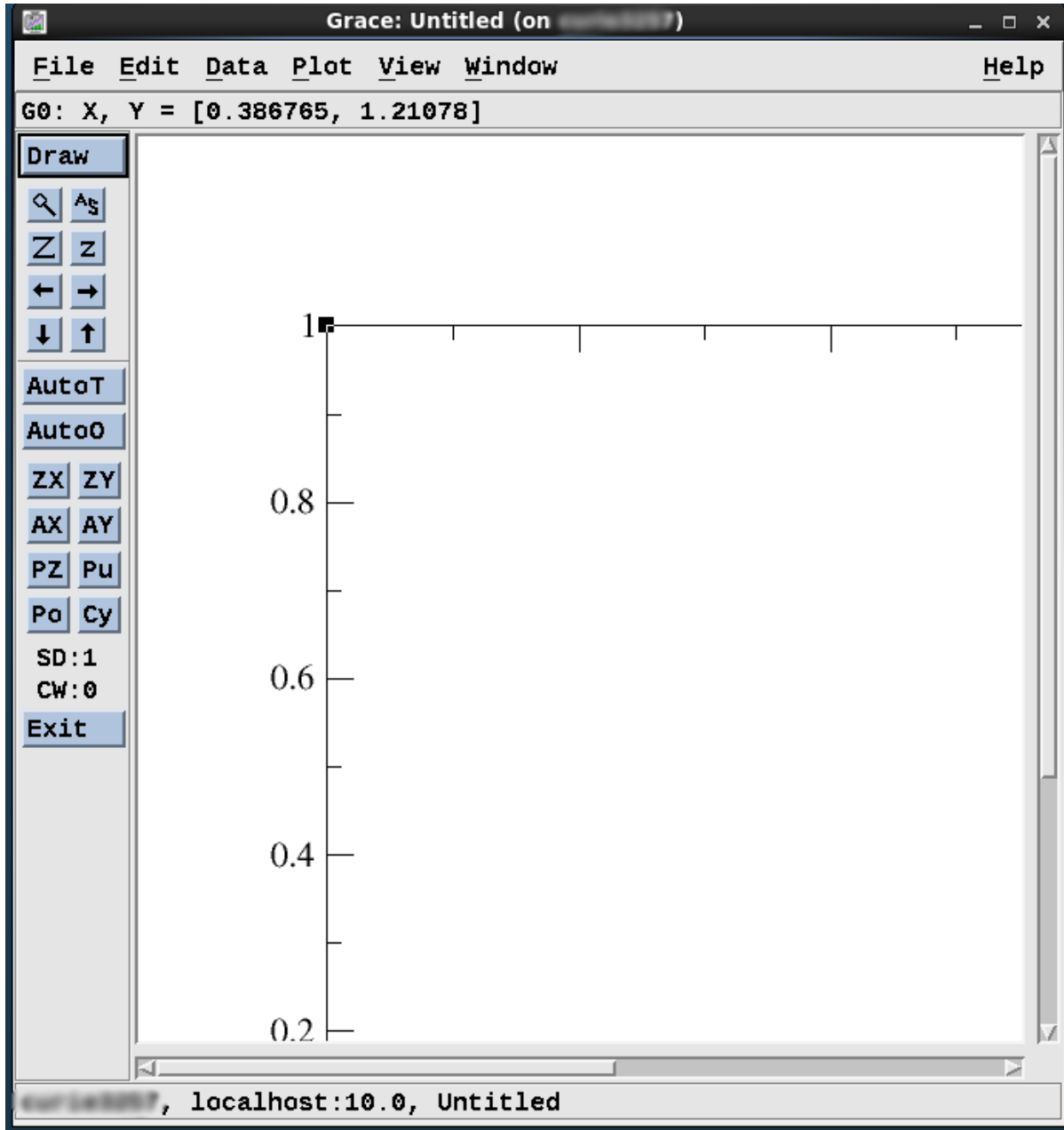
Fig. 2: Example of xmgrace output

```
ccc_mprun -Xfirst -x -Q normal -p <partition> -A <project> tec360
```

Tecplot calls **firefox** to display its help contents that are available as HTML. Since not all node have firefox, you may be interested in the following:

- tutorial

- forum

21.4 Ensight

The tool can be launched from a Remote Desktop System session (highly recommended) or from a compute node. Please avoid login nodes!

From a Remote Desktop System session:

- Start your Remote Desktop System session

- Open a terminal

```
module load ensight
ensight101
```

From a compute node:

- Connect to the supercomputer with trusted X11 forwarding

```
ssh -XY <login>@<machine>
```

- Load ensight

```
module load ensight
```

- Start the tool from a compute node:

```
ccc_mprun -Xfirst -p <partition> -T3600 -A <project> ensight101
```

21.5 visit

21.5.1 Interactive mode

You can use Visit from a Remote Desktop System session or from a compute node (please avoid login nodes)

To start Visit from a Remote Desktop System session:

- Start your Remote Desktop System session

- Open a terminal

```
module load visit
visit
```

To start Visit from a compute node:

- Connect to the supercomputer with trusted X11 forwarding
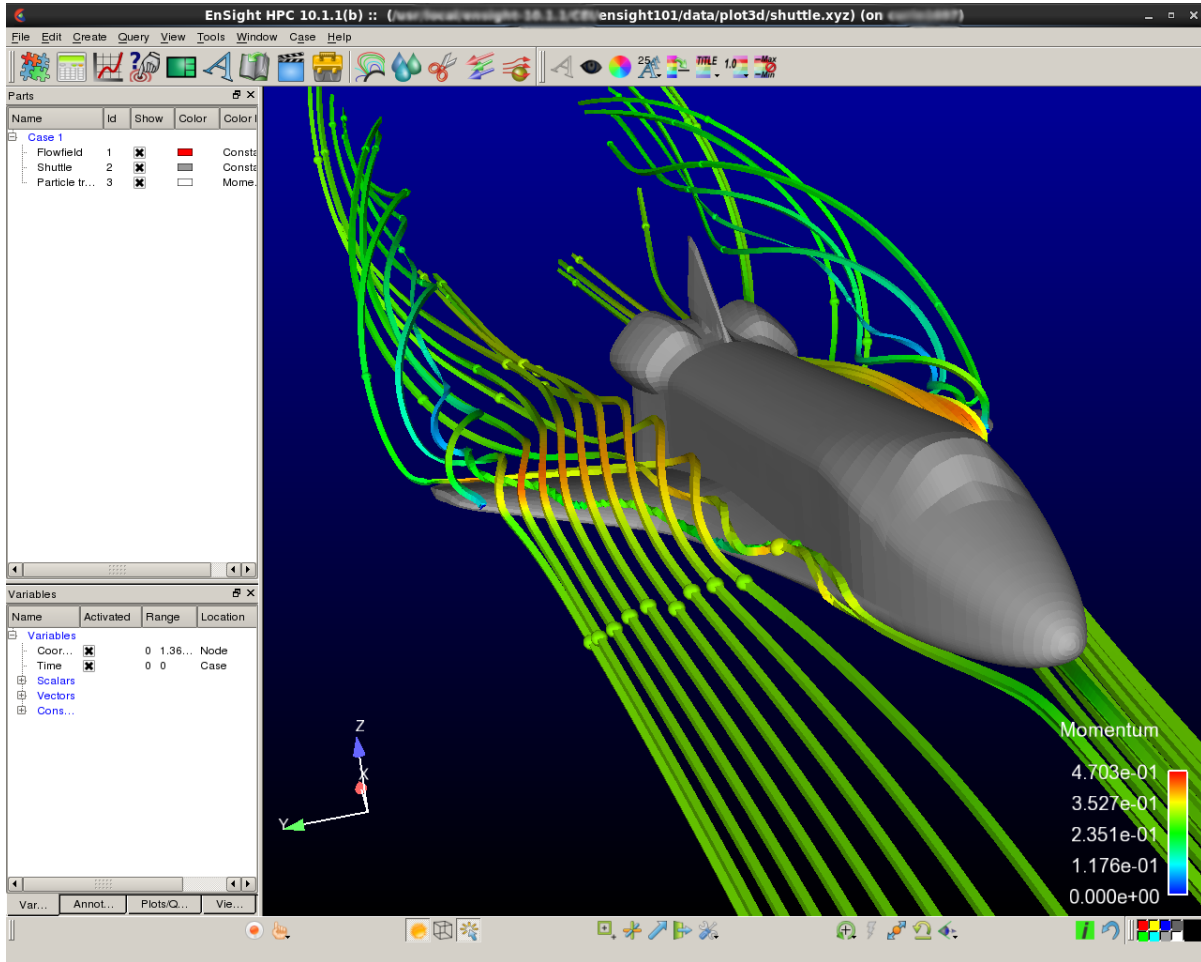
Fig. 3: Example of ensight output

```
ssh -XY <login>@<machine>
```

- Load visit

```
module load visit
```

- Start Visit from a compute node:

```
ccc_mprun -Xfirst -p <partition> -T3600 -A <project> visit -noconfig
```
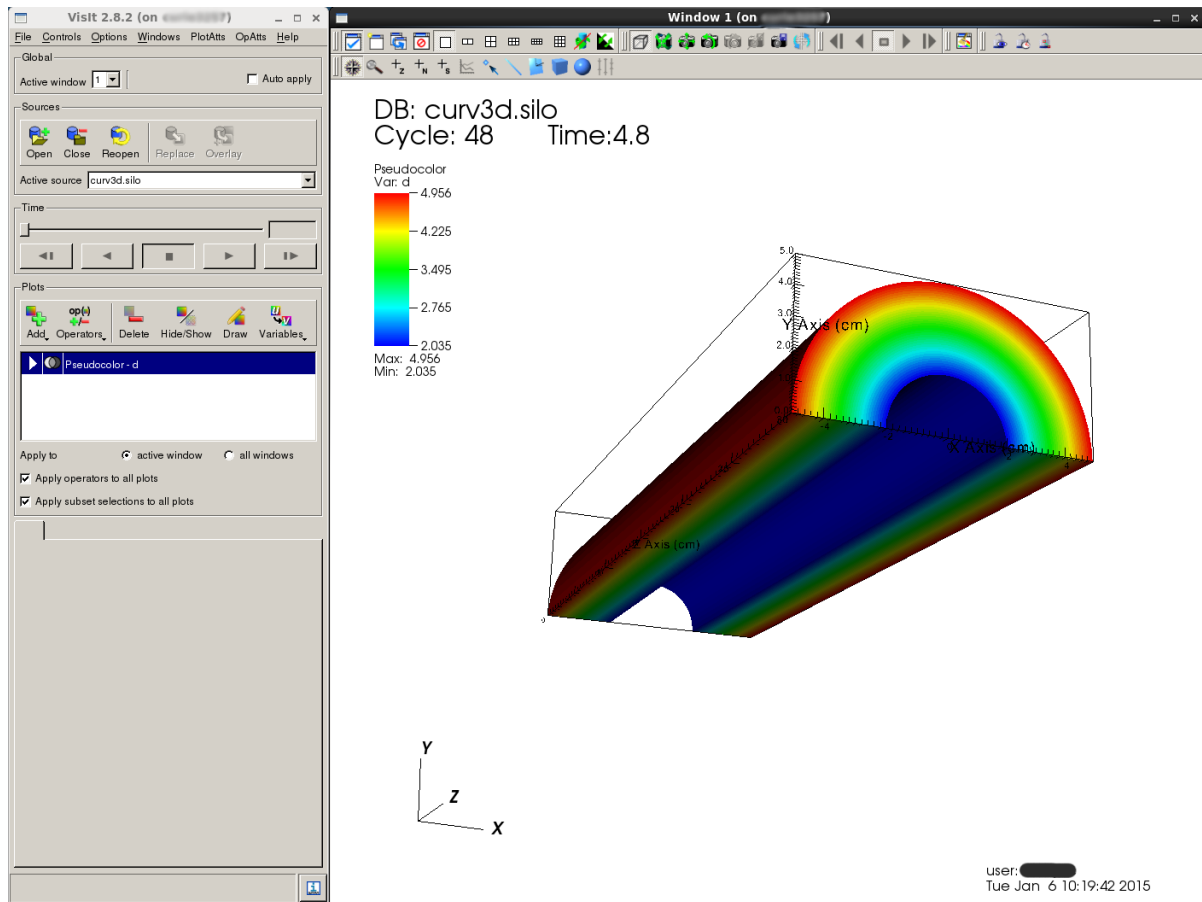

Fig. 4: Example of visit output

21.6 paraview

21.6.1 Interactive mode

You can use Paraview from a Remote Desktop System Session or from a compute node (please avoid login nodes)

To start Paraview from a Remote Desktop System session:

- Start your Remote Desktop System session
- Open a terminal

```
module load paraview
paraview
```

To start Paraview on a compute node:

- Log into the login node, with X redirection and thrusted X11 forwarding

```
ssh -XY login@login-node
```

- Find Paraview:

```
module load paraview
```

- Start Paraview from an exclusive compute node:

```
ccc_mprun -Xfirst -x -Q normal -p <partition> -A <project> paraview
```

21.6.2 Using Python with Paraview

You can use Python with Paraview by using the command **pvpython**. This allows you to load the python module `paraview`.

However to load other modules available in the Python distribution, you will need to load the main Python module. This will link all available modules in the main Python distribution to the **pvpython**.

Note: Each Paraview version is linkable to a single Python version. If **pvpython** is based on Python 3.8 you will need to load the same version using the Python module.

To know which version of Python to load you can use the command below (once Paraview is loaded):

```
pvpython -c 'import sys;print(str(sys.version_info.major) + "." + str(sys.version_info.
→minor))'
```

VIRTUALISATION

22.1 Introduction

Virtualization technologies such as VMs (Virtual Machines) and containers allow users to define custom software environments to facilitate deployment of applications along with all their dependencies, or to help with software development and validation tasks.

On this cluster, the **pcocc** tool (pronounced like peacock) allows to deploy both VMs and containers on compute resources.

Container images built with popular tools such as Docker or Podman can be launched interactively or within Slurm jobs. High performance networks and GPUs can be used from within containers.

Clusters of VMs can be instanciated on the fly with a single command which allocates the necessary resources to host the virtual machines, including private Ethernet and/or InfiniBand networks and starts VMs with ephemeral disks created from a common image.

22.2 Containers

A container is a set of processes which share a custom view of system resources such as the filesystem. Typically, a container is started from an image (a set of files forming a root filesystem) and processes within the container see the image as their root filesystem instead of the filesystem of the host (login or compute node where the container is launched).

A container image can be created using several popular tools (Docker, Podman, . . .) or downloaded from a registry (DockerHub, NVIDIA container registry, . . .). Images are usually distributed in the OCI format which is a standard image format that allows inter-operability between container build tools and runtimes. Another image format is SIF which is mainly used by Singularity (or Apptainer).

Using pcocc, images in the OCI format (limited support is also provided for SIF images) can be imported and run efficiently on a compute cluster. pcocc facilitates integration with the host environment including mounting dataspaces, launching jobs in containers or using *high performance interconnects and GPUs*.

pcocc also provides a way to spawn a *dedicated single-node Docker environment* which allows to work directly with Docker to build containers or run services.

22.2.1 Importing an image and launching containers

1. Generate a tar archive from an OCI image

On your local workstation, build and export a Docker image in the tar archive format:

```
$ docker build -t my_docker_image . # Build an image
$ docker run -ti my_docker_image # Test the image
$ docker save my_docker_image -o my_docker_image.tar # Export the image
```

1. Copy the newly obtained tar file (here: my_docker_image.tar) on the cluster using, for instance, **scp** or **rsync**

2. Import the image in your pcocc image repository on the cluster:

```
$ pcocc-rs image import docker-archive:my_docker_image.tar my_docker_image
```

3. Check that the image is available in your repository

```
$ pcocc-rs image list
```

4. Start a container from your image (once the image has been imported it can be launched multiple times without re-importing)

A single container on the current node:

```
$ pcocc-rs run my_pcocc_image [arg1,arg2,...]
```

Multiple tasks in containers on remote nodes in a batch job:

```
#!/bin/bash
#MSUB -q |default_CPU_partition|
#MSUB -Q test
#MSUB -T 120
#MSUB -n 2
#MSUB -c 2
#MSUB -A <project>
ccc_mprun -n 2 -c 2 -C my_pcocc_image -- <cmd>
```

In that case, one container per node is created and multiple tasks within a node share a container.

When launching a container image with pcocc, there are some notable differences in the execution environment compared to the same image launched using Docker:

- Processes within the container are executed as your user instead of root or the default user specified in the container image.

- A default set of *pcocc modules* is applied which mounts storage spaces (home, scratch, work, store, ...) in the container and propagates a minimal set of common environment variables from the host. Other environment variables are not propagated by default, as when using Docker. The default set of modules may be disabled using the **--no-defaults** option.

- The current working directory is propagated to the container unless it is specified in the image.

- When launching a container through Slurm (**ccc_mprun -C** or **srun --ctr**) the entrypoint and/or command defined when building the image are ignored. However they are executed normally when using **pcocc-rs run**. The entrypoint may be disabled using the **--no-ep** option.

Note: Be careful to load the appropriate **dfldatadir** when using your image in order to have the right **$CCCWORKDIR** definition. As the **pcocc** repository is supposed to be located where the container has been imported, it could be only associated to this specific **dfldatadir**.

22.2.2 Mounting files and directories

Files or directories from the host can be mounted into the container. By default cluster storage spaces (home, scratch, . . .) are mounted in the container on the same location as on the host. This can be disabled using the **--no-defaults** option. Custom mount points may be specified on the command line as follows:

For a local launch:

```
$ pcocc-rs run my_pcocc_image --mount src=<source directory>,dst=<target
↪directory> -- <cmd>
```

For a remote launch using ccc_mprun:

```
$ ccc_mprun -C my_pcocc_image -E'--ctr-mount src=<source directory>,dst=
↪<target directory>' -- <cmd>
```

Alternatively, *modules* or *container templates* can be defined in a configuration file.

22.2.3 Mounting SquashFS images

SquashFS images can also be mounted using a similar syntax:

For a local launch:

```
$ pcocc-rs run my_pcocc_image --mount src=<squashfs_image>,dst=<target directory>,
↪type=squashfs -- <cmd>
```

For a remote launch using ccc_mprun:

```
$ ccc_mprun -C my_pcocc_image -E'--ctr-mount src=<squashfs_image>,dst=<target directory>,
↪type=squashfs' -- <cmd>
```

This can be used in combination with a passthrough container which replicates the host to mount SquashFS images on top of the host filesystem, for example:

```
$ ccc_mprun -C host-passthrough -E'--ctr-mount src=<squashfs_image>,dst=<target
↪directory>,type=squashfs' -- <cmd>
```

22.2.4 Propagating environment variables

Additional environment variables may be propageted from the host to the container using the following syntax:

```
$ export VARIABLE_TEST=valeur
$ pcocc-rs run my_pcocc_image --env VARIABLE_TEST -- <command>
```

Alternatively, *modules* or *container templates* can be defined in a configuration file.

22.2.5 Container modules

Container modules allow to configure a set of host files to mount in a container and of environment variables to propagate in a more convenient way than with command line options.

Modules can be defined in your personal configuration file in ~/.config/pcocc/containers.yaml. Additional configuration files can be provided in *custom locations*. In addition, some modules are provided by the compute center, for exemple to mount storage spaces, *GPU drivers or inject MPI libraries*.

To list available modules:

```
$ pcocc-rs module list
```

Modules listed as default modules are automatically included unless the **--no-defaults** option is specified.

The syntax to define a module is as follows:

```
modules:
  my_module:
    mounts:
      # Mount source_dir_1 on target_dir in the container
      - source: <source_dir__1>
        destination: <target_dir>
      # Mount source_dir_2 on the same location in the container
      - source: <source_dir_2>
    env:
      # Propagate a variable from the host
      - SLURM_JOB_ID
      # Set a variable to a specfic value
      - MY_VARIABLE=<value>
      # Prefix content to a PATH-like variable in the container
      - pp(LD_LIBRARY_PATH)=/opt/mylib
      # Suffix content to a PATH-like variable in the container
      - ps(PATH)=$HOME/mylib
```

To use a module on the command line:

For a local launch:

```
$ pcocc-rs run my_pcocc_image --module my_module  <cmd>
```

For a remote launch using ccc_mprun:

```
$ ccc_mprun -C my_pcocc_image -E'--ctr-module my_module' -- <cmd>
```

22.2.6 Using MPI and NVIDA GPUs in containers

Parallel MPI applications as well as CUDA based applications require low level libraries which talk directly to kernel drivers and/or high performance interconnects and GPUs. These libraries have to match the kernel drivers and hardware of the host.

One way to achieve this is to mount these libraries at runtime using *modules*.

The **nvidia** module mounts low-level libraries needed to access the NVIDIA drivers on compute nodes. For the container to be compatible with the injected libraries it should use a CUDA version that is compatible with the driver version of the compute node (which can be checked with the **nvidia-smi** command) or use the **libcudacompat** library.

```
$ ccc_mprun [usual gpu submission options] -C my_pcocc_image -E '--ctr-module nvidia' --
↪<cmd>
```

The **openmpi-4.1.4** module mounts the OpenMPI library and recommended settings for using the compute nodes high performance interconnect. There is also the module **intelmpi-20.0.0** if you need to run a container with IntelMPI and MPICH. For the container to be compatible with the injected libraries, it should ideally be based on the same OS distribution as the one installed on the host node. More recent Linux distributions should generally work but incompatibilities may exist. The containerized application must be compiled using the same major version of MPI as the one being injected in the container. You must have loaded the MPI environment module with the same version of the choosen MPI pcocc module before running your container. This is required for all MPI libraries to be loaded in your current environment. The **mpi/openmpi** and **mpi/intelmpi** modules define environment variables propagated in the container by the pcocc module.

Example: To run your container with MPI, use the following command:

When you are using OpenMPI:

```
$ module load mpi/openmpi/4.1.4
$ ccc_mprun [usual submission options] -C my_pcocc_image -E '--ctr-module openmpi-4.1.4'
↪-- <cmd>
```

When you are using IntelMPI:

```
$ module load mpi/intelmpi/20
$ ccc_mprun [usual submission options] -C my_pcocc_image -E '--ctr-module intelmpi-20.0.0
↪' -- <cmd>
```

The Nvidia NCCL (NVIDIA Collective Communications Library) which is the multi-GPU communications library can be used in a container with only one of the MPI modules. MPI modules are composed of **nvidia**, **slurm-pmi** and **infiniband** pcocc modules, which are necessary and sufficient for using NCCL in your containers.

To use NCCL, you have two options:

Option 1: Use directly **openmpi-4.1.4** or **intelmpi-20.0.0** pcocc module. Those modules includes both **nvidia**, **slurm-pmi** and **infiniband**, so you don't need to add anything else.

Example: To run your container with an MPI module, use the following command:

```
$ module load mpi/openmpi/4.1.4
$ ccc_mprun [usual gpu submission options] -C my_pcocc_image -E '--ctr-module openmpi-4.
↪1.4' -- <cmd>
```

Option 2: Use **nvidia**, **slurm-pmi** and **infiniband** pcocc modules separately. This approach can be used instead of an MPI module, but remember that MPI modules already contains both **nvidia**, **slurm-pmi** and **infiniband**.

```
$ ccc_mprun [usual gpu submission options] -C my_pcocc_image -E '--ctr-module nvidia,
↪slurm-pmi,infiniband' -- <cmd>
```

Note: Do not use MPI pcocc module with **nvidia**, **slurm-pmi** and **infiniband**. The MPI module already contains **nvidia**, **slurm-pmi** and **infiniband**, using them together can lead to conflicts or errors.

22.2.7 Container templates

A container template combines a container image with runtime launch options to facilitate launching containers which require advanced configuration options.

Container templates can be defined in your personal configuration file in `~/.config/pcocc/containers.yaml`. Additional configuration files can be provided in *custom locations*.

The syntax to define a container template is as follows:

```
containers:
 exemple_template:
   rootfs: "user:my_pcocc_image"
   # Optional. If true, default modules are not loaded
   # (equivalent to pcocc-rs run --no-defaults)
   no_defaults: false
   modules:
     - my_module
   env:
     # Supports the same syntax as modules
     - MY_VARIABLE
   mounts:
     # Supports the same syntax as modules
     - source: <source_dir_1>
       destination: <target_dir>
```

A container template can reference *modules* or define environment variables and mounts inline. The rootfs field defines the container image to use. In this example the my_pcocc_image image in the 'user' repository is selected.

The template name can be passed to **pcocc-rs run** or **ccc_mprun -C** instead of a container image name.

For a local launch:

```
$ pcocc-rs run my_template -- <cmd>
```

For a remote launch using ccc_mprun:

```
$ ccc_mprun -C my_template <cmd>
```

22.2.8 Configuration files and image repository locations

By default, images and container configurations (modules, templates, ...) are sourced from the per-user configuration directory and image repository. The per-user configuration directory is located at ~/.config/pcocc/. Container configuration files for *modules* and *templates* can be defined in ~/.config/pcocc/containers.yaml and ~/.config/pcocc/containers.d/*.yaml.

Additional image repositories can be defined in ~/.config/pcocc/repositories.yaml and ~/.config/pcocc/repositories.d/*.yaml.

The syntax to define a repository is as follows:

```
repositories:
  myrepo:
    path: </path/to/my/repo>
    description: This is my repository
```

A new repository is automatically initialized if the path is an empty directory or if it doesn't exist (the parent directory must however exist).

To use an image from a non-default repository, the repository name must be prefixed to the image name, for example:

```
$ pcocc-rs run myrepo:myimage
```

To list images from a specific repository use the `--repo` option, for example:

```
$ pcocc-rs image list --repo myrepo
```

Additional configuration directories can be defined by setting the `PCOCC_CONFIG_PATH=<dir1>[:<dirN>]` variable. In each additional configuration directory, the same configuration files as in the per-user configuration directory can be set. This can be combined with shared spaces and extenv modules to provide container definitions to a group of users.

22.2.9 Importing SIF images

SIF images can be imported using:

```
$ pcocc-rs image import sif:<mysifimage.sif> mysifimage
```

Please note however, that environment variables defined in SIF images may not be fully preserved when converted for use with `pcocc-rs`.

22.2.10 Migrating from pcocc to pcocc-rs

`pcocc-rs` is a new version of pcocc written in Rust which is now recommended for launching containers. While the `pcocc` command is still available, it should no longer be used for that purpose.

Image repositories are shared between `pcocc` and `pcocc-rs` but if you imported images with the `pcocc` command, they will at first appear with a forbidden sign in the output of `pcocc-rs image list`. In that case, you have to rebuild them for pcocc-rs using the following command before running them:

```
$ pcocc-rs image rebuild my_docker_image
```

The "No squashfs has been built for this image" error message usually indicates that `pcocc-rs`` is trying to start a `pcocc` image that hasn't been rebuilt yet.

22.3 Docker environment

pcocc allows to create a dedicated, single-node, virtual Docker environment. Inside this environment, Docker commands can be used as normal to build and test Docker containers. Behind the scenes, a Docker daemon is started in a VM and handles the requests from the Docker CLI as if the daemon was running locally. Once containers have been built in this environment they can be exported to a pcocc image repository to start them directly on login or compute nodes without using a Docker daemon.

This Docker environment should only be used to build or develop with containers, or for workflows that are tightly integrated with Docker. It only scales to a single node and induces substantial overhead on I/O operations when mounting data from the host cluster in containers.

To run containerized applications efficiently, containers should be started directly using pcocc-rs instead of using Docker (see the *Containers* section).

In any case, access to external resources (such as public image registries, package or source code repositories, etc.) is only possible if the host cluster can access the Internet.

22.3.1 Starting a Docker environment interactively

To start an interactive Docker session:

```
$ pcocc docker alloc -p <partition> -c <number of cores>
salloc: Pending job allocation XXXXXX
salloc: job XXXXXX queued and waiting for resources
salloc: job XXXXXX has been allocated resources
salloc: Granted job allocation XXXXXX
salloc: Waiting for resource configuration
salloc: Nodes node6020 are ready for job
Configuring hosts... (done)
Waiting for Docker VM to start ...
(pcocc/XXXXXX) $ docker --version
Docker version XX.YY.ZZ, build xxxxxxx
(pcocc/XXXXXX) $ exit
$
```

In some cases you may be asked for an account which can be specified with **-A <account>**. Please ask the hotline if you don't know which account to use.

Please note that by default an ephemeral VM is used for the Docker daemon where all containers and images are stored in memory. The number of allocated cores determines the amount of memory available for Docker to store containers. Alternatively, the docker Daemon can also be *configured to use persistent data*.

In the Docker session, Docker commands can be used as normal. However, access to external resources is only possible on clusters where Internet access is authorized:

```
(pcocc/XXXXXX) $ docker search ubuntu
(pcocc/XXXXXX) $ docker pull ubuntu:latest
NAME   DESCRIPTION                              STARS OFFICIAL AUTOMATED
ubuntu Ubuntu is a Debian-based Linux operating sys... 15605 [OK]
(pcocc/XXXXXX) $ docker image list
REPOSITORY    TAG        IMAGE ID       CREATED          SIZE
ubuntu        latest     58db3edaf2be   3 weeks ago      77.8MB
(pcocc/XXXXXX) $ docker run -ti ubuntu:latest bash
root@c02d04424d02:/# cat /etc/debian_version
```

(continues on next page)

```
bookworm/sid
(pcocc/XXXXXX) $ docker run -ti ubuntu:latest bash
(pcocc/XXXXXX) $ cat Dockerfile
FROM ubuntu:latest
RUN apt update && apt upgrade -y
(pcocc/XXXXXX) $ docker build -t myubuntu:latest .
(pcocc/XXXXXX) $ docker image list
REPOSITORY    TAG       IMAGE ID      CREATED             SIZE
myubuntu      latest    c56a83f1ec3a  About a minute ago  128MB
ubuntu        latest    58db3edaf2be  3 weeks ago         77.8MB
(pcocc/XXXXXX) $ docker save myubuntu:latest -o myubuntu_latest.tar
```

At the end of a Docker session, the whole state of the Docker daemon, including all container and container images is destroyed. Container images can be exported to a pcocc image repository to persist them and/or launch containers on the cluster without using Docker as described in the *Containers* section:

```
(pcocc/XXXXXX) $ pcocc-rs image import docker-daemon:myubuntu:latest myubuntu
Successfully imported docker-daemon:myubuntu:latest
(pcocc/XXXXXX) $ exit
$ pcocc-rs image list
Name                         Type         Modified
========================================================
myubuntu                     container    20XX-02-16T18:11:15
```

To retrieve the Docker image from a pcocc image repository in a future session:

```
$ pcocc docker alloc -p <partition> -c <number of cores>
(pcocc/XXXXXX) $ pcocc docker import user:myubuntu myubuntu:latest
(pcocc/XXXXXX) $ docker image list
REPOSITORY    TAG       IMAGE ID      CREATED             SIZE
myubuntu      latest    1f382964a2b1  About an hour ago   128MB
```

22.3.2 Persisting the Docker daemon state across allocations

Docker sessions can be persisted across successive allocations using the persistent Docker template. First, the persistent environment must be initialized once by creating a persistent disk image for the VM hosting the Docker daemon:

```
$ qemu-img create -f raw $CCCWORKDIR/.pcocc.docker.persistent.img 100G
$ mkfs.ext4 $CCCWORKDIR/.pcocc.docker.persistent.img
```

To make use of this disk image, set the option **-t docker-persist** when launching a Docker session:

```
$ pcocc docker alloc -t docker-persist -p <partition> -c <number of cores>
```

Please be aware that in this mode, a single Docker daemon can be allocated at a time since this data disk cannot be shared by multiple daemons.

22.3.3 Submitting jobs in a Docker environment

The `pcocc docker batch` command can be used to submit a batch job which can execute Docker commands.

```
$ pcocc docker batch -p <partition> -c <number of cores>
```

Once the Docker daemon is running in batch mode the Docker environment can be joined using `pcocc docker shell`.

A build script with docker commands can be used in both interactive or batch mode

For example, given this script:

```
$ cat docker_job.sh
#!/usr/bin/bash
docker build -t myubuntu:latest .
docker save myubuntu:latest -o myubuntu_latest.tar
```

It can be run with:

```
$ pcocc docker alloc -E 'docker_job.sh'
```

or

```
$ pcocc docker batch -E 'docker_job.sh'
```

22.4 Virtual machines

pcocc allows to run clusters of VMs on compute nodes, alongside regular jobs.

To launch a virtual cluster, a user selects one or more templates from which to instantiate VMs and the number of requested VMs. For each virtual cluster, pcocc allocates the necessary resources to host the VMs, including private Ethernet and/or Infiniband networks, creates temporary disk images from the selected templates and instantiates the requested VMs.The VMs are ephemeral and use Copy-on-Write disk drives instanciated using the template image as a backing file.

Full documentation is provided through pcocc man pages. Use `man pcocc` for an index and refer to the individual man pages listed at the bottom the page for more specific information for example: `man pcocc-templates.yaml`. The documentation is also available on the web: <*https://pcocc.readthedocs.io/en/latest*>

The following sections provide a quick overview of how some common tasks can be performed.

22.4.1 Building or importing images

Any disk image in a file format supported by Qemu can be imported using:

```
$ pcocc image import /path/to/my/image user:[image name]
```

Depending on the image format, and filename extension, the file format may need to be specified using the `--fmt` option.

There are no particular requirements on images except that DHCP should be enabled on the main interface to configure the network automatically. It's also useful that images are configured to use the serial console. Vanilla cloud images from Linux distributions are usually well suited as a base for creating images that work well with pcocc.

To manage images and image repositories, please refer to the dedicated man page with `man pcocc-image`.

Cloud images can also be configured by pcocc using cloud-init. Please refer to the vm creation tutorial (**man pcocc-newvm-tutorial**).

22.4.2 Launching a virtual cluster

Once an image has been imported, a cluster of VMs can be instanciated from it. More precisely, VMs are instanciated from templates which refer to an image and other configuration parameters such as network resources or host files to expose to the VM. However, for each image, a template with the same name is implicitely created. It allows to instanciate a VM from the image with a default configuration.

The default network is a private Ethernet network interconnecting all VMs of virtual a cluster. It features a virtual router providing access to the host cluster from the VM private network using NAT. It also ensures that VMs are reachable through SSH from the host cluster using reverse NAT.

For more information about VM templates, please refer to **man pcocc-templates.yaml**

To start a cluster of VMs from VM images:

```
$ pcocc alloc -p <partition> -c <cores per VM>  myimage1:<number of VMs>,myimage2:
→<number of VMs>
```

The boot of a VM can usually be monitored through its console (hit Ctrl-C 3 times to exit the console):

```
(pcocc/1016696) $ pcocc console vm0
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[...]
```

If SSH access has been properly configured, VMs should be reachable using SSH:

```
(pcocc/1016696) $ pcocc ssh vm0
```

For more information about launching VMs with pcocc, please refer to the main man page **man pcocc** and to the man pages for each subcommand for example **man pcocc-alloc**. Help is also available for each subcommand, for example **pcocc batch --help**. The documentation is also available on the web: *<https://pcocc.readthedocs.io/en/latest>*

QUANTUM SOFTWARE STACK

23.1 Quantum computing at TGCC

Quantum computing at TGCC is made available through an offer of hardware, today including emulators and simulators, and a comprehensive software stack.

This quantum service can be used to emulate quantum problems on cluster nodes or on Qaptiva emulator, and to execute quantum programs on Pasqal QPU.

On the hardware side, apart from compute nodes that can be used to emulate quantum systems, we provide a Qaptiva emulator (a dedicated hardware to quantum emulation) as well as a physical QPUs from Pasqal.

To run quantum algorithms, we provide a comprehensive software stack, notably including myQLM (with many features from digital to analog quantum computing, and to target Qaptiva), Pulser (to emulate and use Pasqal hardware) and Perceval (to emulate Quandela hardware). This software environment is set up in a container image to ease its use (`ccc-quantum`).

23.2 Connection and environment

The quantum software stack is provided in a container image called `ccc-quantum` which can be started on a login node or interactively on a compute node with the command below:

```
pcocc-rs run ccc-quantum
```

It is possible to launch directly this image on a compute node with the following command, or to submit a batch script containing this command:

```
ccc_mprun -C ccc-quantum -p <partition> -c <number of cores> python3 <script.py>
```

Note: If you need to emulate large quantum problems outside of Qaptiva, it is strongly advised to work on a compute node and not on the login nodes that are not designed to launch computation and have limited resources.

Note: All quantum computing resources are accessed via the `ccc-quantum` container.

`ccc-quantum` container image provides a series of quantum computing libraries mandatory to use the available hardware as well as popular quantum computing libraries. This list of libraries includes, but is not limited to:

- myQLM

- Pulser and Pulser-myQLM binding

- Perceval

This environment also includes tutorials that can all be found in **/opt/tutorials**. It is strongly advised to copy locally (on your **$HOME** directory or on your **$CCCWORKDIR**) to use these tutorials and keep your modifications.

23.3 Graphical Environment

To launch Jupyter notebooks, first allocate a remote desktop environment on a visualization node using **ccc_visu**. To access the cluster, please refer to the *Interactive access section*. For example:

```
$ ccc_visu virtual -p partition
```

Once logged in to the remote desktop environment, launch a terminal and start the container:

```
$ pcocc-rs run ccc-quantum
```

Within the container, copy the training notebooks to your home directory, for example:

```
$ cp -r /opt/tutorials/myqlm $HOME/myqlm_notebooks
```

Launch jupyter (a firefox window will open):

```
$ cd $HOME/myqlm_notebooks
$ jupyter notebook
```

Note: Jupyter notebooks dedicated to training are available in the **ccc-quantum** container.

Once you are comfortable with writing software using myQLM, you can submit your quantum jobs to a Qaptiva machine instead of running them locally. The Qaptiva devices are optimized for running programs with a large number of qubits. The connection settings has to be skipped as the configuration is already done in the **ccc-quantum** container.

23.4 One environment, different use cases

The quantum software stack provided within **ccc-quantum** can be used to emulate both digital (gate-based) and analog quantum systems.

- By using the free API of myQLM or the default quantum computing libraries provided (Pulser, Perceval), it is possible to emulate locally on a node a quantum system. In this case, it is strongly advised to use a compute node, for a better usage of TGCC computing resources.

- By using Qaptiva Access or a dedicated library binding, it is possible to emulate quantum system by harnessing the computing power of the Qaptiva appliance. Qaptiva appliance both offers dedicated hardware with lots of resources (namely, memory) to emulate quantum systems and optimized functions yelding faster computations.

- Lastly, Qaptiva Access can also be used to submit jobs on physical QPUs.

Each of these use cases will be discussed in the following sections.

Note:

- myQLM is an open-source quantum software stack for quantum programs, provided by Eviden.

- QLM is the former name of the hardware dedicated to running quantum emulation.

- Qaptiva now encompass the hardware and software appliances dedicated to quantum emulation.

- Qaptiva Access (also known as QLMaaS) is the interface users can queue quantum jobs with on Qaptiva hardware or on physical QPUs.

23.5 Examples: basic programs

The following examples will show how to use the provided software stack on a compute node.

To launch tutorials provided in `/opt/tutorial` please refer to the *Graphical Environment* section.

23.5.1 myQLM

This simple program explores a simple case of digital (gate-based) quantum programming, with the application of multiple Hadamard (H) and CNOT gates using myQLM on a qubit state. The theoretical probability is 0.25 for the state 000, 001, 110 and 111.

```python
from qat.lang.AQASM import Program, H, CNOT

# Create a Program
qprog = Program()
# Number of qubits
nbqubits = 3
# Allocate some qubits
qubits = qprog.qalloc(nbqubits)

# Apply some quantum Gates
qprog.apply(H, qubits[0])
qprog.apply(CNOT, qubits[0], qubits[1])
qprog.apply(CNOT, qubits[1], qubits[2])
qprog.apply(H, qubits[2])
# Export this program into a quantum circuit
circuit = qprog.to_circ()
circuit.display()

# Import one Quantum Processor Unit Factory
from qat.qpus import PyLinalg

# Create a Quantum Processor Unit
qpu = PyLinalg()

# Create job and submit it
job = circuit.to_job(nbshots=10000)
result = qpu.submit(job)

# Print results
for sample in result:
    print("The state {} have a probability measured equal to {}".format(sample.state,
→sample.probability))
```

As you can see, increasing the number of shots tends towards the theoretical result.

More example programs are provided in the official QLM documentation. Official tutorials from myQLM are also available directly within `ccc-quantum` container image at `/opt/tutorials/myqlm`.

23.5.2 Pulser

This is a simple example of analog computing with Pulser. This example, adapted from the official Pulser documentation, shows how to create an antiferromagnetic state.

```python
import numpy as np
import pulser
import qutip
from pulser_simulation import QutipEmulator

# Setup
L = 8
Omega_max = 2.3 * 2 * np.pi
U = Omega_max / 2.3
delta_0 = -3 * U
delta_f = 1 * U
t_rise = 2000
t_fall = 2000
t_sweep = (delta_f - delta_0) / (2 * np.pi * 10) * 5000

# Define a register: a ring of atoms distanced by a blockade radius distance:
R_inter = pulser.MockDevice.rydberg_blockade_radius(U)
coords = (R_interatomic / (2 * np.tan(np.pi / L))
        * np.array([
            (np.cos(theta * 2 * np.pi / L), np.sin(theta * 2 * np.pi / L))
            for theta in range(L)]))

reg = pulser.Register.from_coordinates(coords, prefix="atom")
reg.draw(blockade_radius=R_inter, draw_half_radius=True, draw_graph=True)

# Define a pulse sequence
rise = pulser.Pulse.ConstantDetuning(pulser.RampWaveform(t_rise, 0.0, Omega_max), delta_
↪0, 0.0)
sweep = pulser.Pulse.ConstantAmplitude(Omega_max, pulser.RampWaveform(t_sweep, delta_0,
↪delta_f), 0.0)
fall = pulser.Pulse.ConstantDetuning(pulser.RampWaveform(t_fall, Omega_max, 0.0), delta_
↪f, 0.0)

seq = pulser.Sequence(reg, pulser.MockDevice)
seq.declare_channel("ising", "rydberg_global")

seq.add(rise, "ising")
seq.add(sweep, "ising")
seq.add(fall, "ising")
seq.draw()

# Emulate the system locally
sim = QutipEmulator.from_sequence(seq, sampling_rate=0.1)
```

```
results = sim.run(progress_bar=True)

# Retrieve the results
n_samples = 1000
counts = results.sample_final_state(n_samples)

large_counts = {k: v for k,v in counts.items() if v > 5}
```

Examples can be found in the official Pulser documentation. The official tutorials (from the GitHub repository) are also available at **/opt/tutorials/pulser/**.

23.5.3 Perceval

Examples can be found in the official Perceval documentation. The official tutorials (from the GitHub repository) are also available at **/opt/tutorials/perceval/**.

23.6 Using Qaptiva

23.6.1 From compute nodes to Qaptiva

Using myQLM you can access libraries and tools dedicated to quantum programming. These programs will use **pyLinalg** as the QPU, a Python-based linear algebra library. Jobs can be submitted on any regular compute node.

Using Qaptiva Access, you will be able to use the Qaptiva hardware appliance, which is also a gateway to physical QPUs. Qaptiva hardware appliance can emulate up to 40 exact qubits. It has an internal queueing system, and all jobs submitted to the appliance will be executed asynchronously. Its hardware specifications are available below:

- 24 TB of RAM
- 16 A30 GPU with 24GB of vRAM
- 384 Intel Cascade Lake 2.4GHz

The same goes for Pulser; Qutip emulation will be used on a compute node but the capabilities of Qaptiva hardware appliance can be used.

Note: We recommend using myQLM on a compute node to learn about quantum computing. Once you feel confortable with the concepts, Qaptiva allows to emulate more qubits. Then, real simulation can be run using the Pasqal machine (Pasqal QPU).

23.6.2 Digital quantum computing

To use Qaptiva appliance in the Qaptiva Access mode, you simply have to change the emulated QPU used, from **qat.qpus.PyLinalg** to **qlmaas.qpus.LinAlg**. Since jobs are executed asynchronously, their completion has to be checked before actually using the results.

The example provided in myQLM example can be rewritten:

```python
from qat.lang.AQASM import Program, H, CNOT

# Create a Program
qprog = Program()
# Number of qubits
nbqubits = 3
# Allocate some qubits
qubits = qprog.qalloc(nbqubits)

# Apply some quantum Gates
qprog.apply(H, qubits[0])
qprog.apply(CNOT, qubits[0], qubits[1])
qprog.apply(CNOT, qubits[1], qubits[2])
qprog.apply(H, qubits[2])
# Export this program into a quantum circuit
circuit = qprog.to_circ()
circuit.display()

# Import one Quantum Processor Unit Factory
from qlmaas.qpus import LinAlg

# Create a Quantum Processor Unit
qpu = LinAlg()

# Create job and submit it
job = circuit.to_job(nbshots=10000)
asynchronous_result = qpu.submit(job)

# Wait for the job to complete
result = asynchronous_result.join()

# Print results
for sample in result:
    print("The state {} have a probability measured equal to {}".format(sample.state,
→sample.probability))
```

Using GPUs

As Qaptiva appliance comes with GPUs, it is possible to use this units to accelerate the emulation. the `qlmaas.qpus.LinAlg` constructor takes an argument `use_GPU` that can be set to `True` to be executed on GPU. The default is `False`.

More details can be found in the dedicated documentation on GPU acceleration feature in QLM.

23.6.3 Pulser

It is possible to emulate quantum system designed with Pulser on Qaptiva. To do so, you need to use:

- **AnalogQPU** from myQLM, that allows to emulate an analog QPUs

- **IsingAQPU** from Pulser-myQLM binding, that translates a Pulser quantum system to an object that can be used by Qaptiva.

The quantum system will then be executed on Qaptiva and the result will be a myQLM object.

The last part of the example provided in Pulser example can be rewritten as follows:

```python
from pulser_myqlm import IsingAQPU
from qlmaas.qpus import AnalogQPU

# Create an analog QPU, convert the sequence to a Qaptiva job and run
aqpu = AnalogQPU(n_step=1000)
qpu = IsingAQPU.from_sequence(seq, qpu=aqpu)
job = IsingAQPU.convert_sequence_to_job(seq, nbshots=0)
res = qpu.submit(job)

# Retrieve the results
dict_res = {sample.state: sample.probability for sample in res if sample.probability >␣
→0005}
```

More examples can be found in the official tutorials (from the GitHub repository), also available at **/opt/tutorials/ pulser-myqlm/**.