UG3.3: GALILEO100 UserGuide

In this page:

---

hostname: login.g100.cineca.it

login01-ext.g100.cineca.it

login02-ext.g100.cineca.it

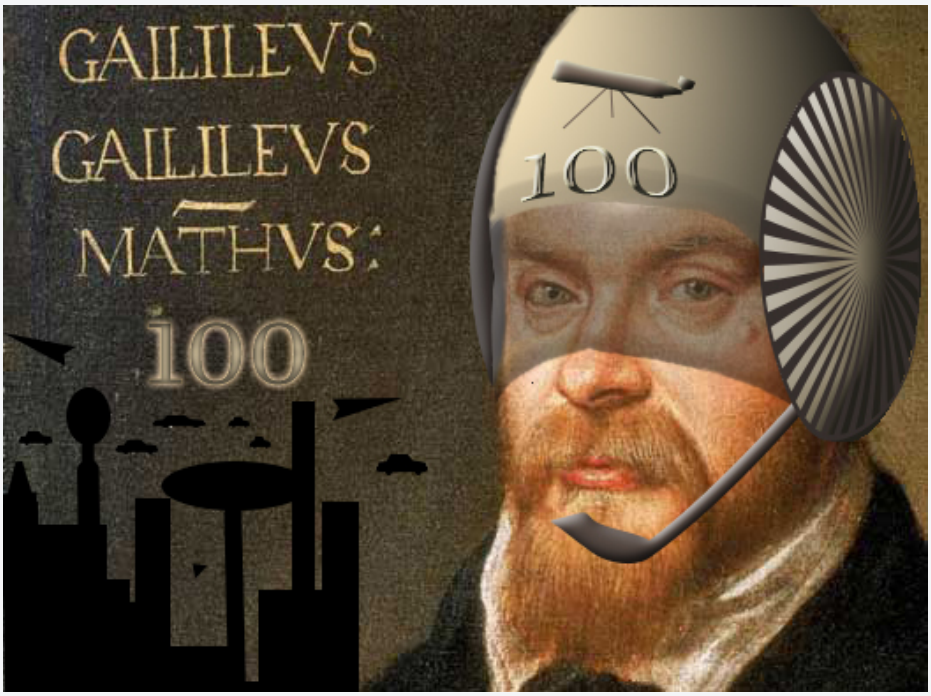
login03-ext.g100.cineca.it

early availability: 20/07/2021

start of pre-production: 09 Aug 2021

start of production: 15 Oct 2021

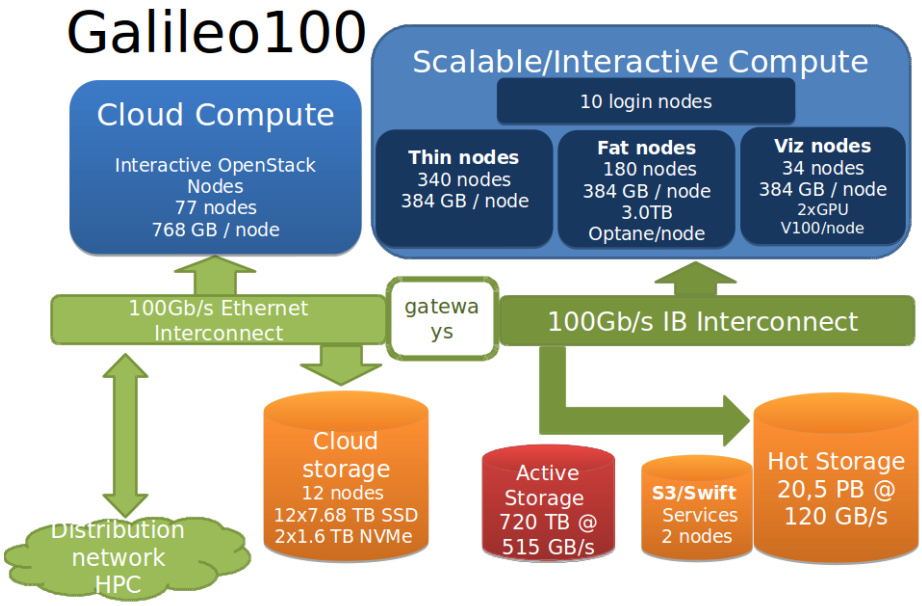
```
Model: DUal-Socket Dell
PowerEdge
Architecture: Linux
Infiniband Cluster
Nodes:  636 (+10 login nodes)
Processors:2xCPU x86 Intel
Xeon Platinum 8276-8276L
          (24c, 2.4Ghz)

Cores:48 cores/node
Accelerators: 2xGPU nVidia
V100 PCIe3 with 32GB RAM on
36 Viz nodes
RAM: 384GB (+ 3.0TB Optane on
180 fat nodes)
Internal Network: Mellanox
Infiniband 100GbE
Peak performance single node:
3.53 TFlop/s
Peak performance - total :
about 2 PFlop/s
```



Starting from **March 2021** Galileo was turned off to make space for the new Galileo100.
The new Infrastructure is co-funded by the European ICEI (Interactive Computing e-Infrastructure) project, it is a system engineered by DELL.

Starting from **November 2022** two additional rack for a total of **82 "thin nodes"** were added to the cluster.

System Architecture



Compute Nodes:

- 636 computing nodes each 2 x CPU Intel CascadeLake 8260, with 24 cores each, 2.4 GHz, 384GB RAM, subdivided in:
  - 422 standard nodes ("thin nodes") 480 GB SSD
  - 180 data processing nodes ("fat nodes") 2TB SSD, 3TB Intel Optane
  - 34 GPU nodes (visualization "viz") with 2x NVIDIA GPU V100 with 100Gbs Infiniband interconnection and 2TB SSD.
- 77 computing servers OpenStack for cloud computing (ADA CLOUD), 2x CPU 8260 Intel CascadeLake, 24 cores, 2.4 GHz, 768 GB RAM, with 100Gbs Ethernet interconnection.
- 20 PB of active storage accessible from both cloud and HPC nodes.
- 1 PB Ceph storage for Cloud (full NVMe/SSD)
- 720 TB fast storage (IME DDN solution)

Login and Service nodes:

10 login nodes and 5 service nodes. All the nodes are interconnected through an Infiniband network, with OPA v10.6, capable of a maximum bandwidth of 100Gbit/s between each pair of nodes.

Access

All the login nodes have an identical environment and can be reached with SSH (Secure Shell) protocol using the "collective" hostname:

> login.g100.cineca.it

which establishes a connection to one of the available login nodes. To connect to Galileo100 you can also indicate explicitly the login nodes:

> login01-ext.g100.cineca.it
> login02-ext.g100.cineca.it
> login02-ext.g100.cineca.it

For information about data transfer from other computers please follow the instructions and caveats on the dedicated section **Data storage** or the document Data Management.

Accounting

For more information about accounting, please consult our **dedicated section**.

Budget Linearization policy

On GALILEO100 a linearization policy for the usage of project budgets has been defined and implemented. For each account, a monthly quota is defined as:

```
monthTotal = (total_budget / total_no_of_months)
```

Starting from the first day of each month, the collaborators of any account are allowed to use the quota at full priority. As long as the budget is consumed, the jobs submitted from the account will gradually lose priority, until the monthly budget (monthTotal) is fully consumed. At that moment, their jobs will still be considered for execution, but with a lower priority than the jobs from accounts that still have some monthly quota left.

This policy is similar to those already applied by other important HPC centers in Europe and worldwide. The goal is to improve the response time, giving users the opportunity of using the cpu hours assigned to their project in relation to their actual size (total amount of core-hours).

Disks and Filesystems

The storage organization conforms to the CINECA infrastructure (see Section "Data storage and Filesystems") . In addition to the home directory (**$HOME**), for each user is defined a scratch area **$CINECA_SCRATCH**, a large disk for storing run time data and files. The new variable **$SCRATCH** is also available, resolving in the same path of $CINECA_SCRATCH. **$WORK** is defined for each active project on the system, reserved for all the collaborators of the project. This is a safe storage area to keep run time data for the whole life of the project.

The filesystem organization is based on LUSTRE open source parallel file system.

	Total Dimension (TB)	**Quota (GB)**	**Notes**
$HOME	100T	50 GB quota per user	• permanent/backed up, user specific, local
$SCRATCH (on G100)	1PB	no quota	• temporary, user specific, local • automatic cleaning procedure of data older than 40 days (time interval can be reduced in case of critical usage ratio of the area. In this case, users will be notified via HPC-News)
$WORK	2PB	1TB quota per project	• permanent, project specific, local • extensions can be considered if needed (mailto: superc@cineca.it)

It is also available a temporary storage local on compute nodes generated when the job starts and accessible via environment variable **$TMPDIR**. For more details please see the dedicated section of UG2.5: Data storage and FileSystems. On Galileo100 the $TMPDIR local area has 293 GB of available space.

$DRES points to the shared repository where **D**ata **RES**ources are maintained. This is a data archive area **available only on-request,** shared with all CINECA HPC systems and among different projects.

$DRES is not mounted on the compute nodes. This means that you **can't access it within a batch job**: all data needed during the batch execution has to be moved on $WORK or $CINECA_SCRATCH before the run starts.

Use the local command "cindata" to query for disk usage and quota ("cindata -h" for help):

```
> cindata
```

or the tool "cinQuota" available in the module cintools

```
> cinQuota
```

For more details about both these commands, please consult the section dedicated to how to monitor the occupancy

Dedicated node for Data transfer and download

A time limit of 10 cpu-minutes for processes running on login nodes has been set.
For Data transfer or download that may require more time, we set up a dedicated "data" VM accessible with a dedicated alias.
Login via ssh to this VM is not allowed. Environment variables as $HOME or $WORK are not defined, so you always have to make the complete path to the files you need to copy explicit.
For example to copy data to Galileo100 using **rsync** you can run the following command:

```
rsync -PravzHS </data_path_from/file> <your_username>@data.g100.cineca.it:<complete_data_path_to>
```

You can also use the "data" VM onto login nodes to move data from Galileo100 to another location with public IP:

```
ssh -xt data.g100.cineca.it rsync -PravzHS <complete_data_path_from/file> </data_path_to>
```

this command will open a session on the VM that will not be closed until the rsync command is completed.

In similar ways you can use also scp and sftp commands if you prefer them.

The data VM also offers **wget** and **curl** commands. If you need to wget a large amount of data for a public site, you can run the following command:

```
ssh -xt data.g100.cineca.it wget <url/file> -P </data_path_to>
```

this command will open a session on the VM that will not be closed until the wget command is completed.

Modules environment

The software modules are collected in different profiles and organized by functional category (compilers, libraries, tools, applications,..).

On GALILEO100 the profiles are of two types, “domain” type (bioinf, chem-phys, lifesc,..) for the production activity and “programming” type (base and advanced) for compilation, debugging and profiling activities and that they can be loaded together.

"Base" profile is the default. It is automatically loaded after login and it contains basic modules for the programming activities (intel e gnu compilers, math libraries, profiling and debugging tools,..).

If you want to use a module placed under other profiles, for example an application module, you will have to load preventively the corresponding profile:

```
>module load profile/<profile name>
```

```
>module load autoload <module name>
```

For listing all profiles you have loaded you can use the following command:

```
>module list
```

In order to detect all profiles, categories and modules available on GALILEO100 the command “modmap” is available:

```
>modmap
```

With modmap you can see if the desired module is available and which profile you have to load to use it.

```
>modmap -m <module name>
```

Spack environment

In case you don't find a software you are interested in, you can install it by yourself.
In this case, on GALILEO100 we also offer the possibility to use the “spack” environment by loading the corresponding module. Please refer to the dedicated section in UG2.6: Production Environment.

Production environment

Since GALILEO100 is a general purpose system and it is used by several users at the same time, long production jobs must be submitted using a *queuing system*. This guarantees that access to the resources is as fair as possible.

Roughly speaking, there are two different modes to use an HPC system: **Interactive** and **Batch**. For a general discussion see the section "Production Environment".

Interactive

A **serial program** can be executed in the standard UNIX way:

```
> ./program
```

This is allowed only for **very short** runs, since the interactive environment set on the login nodes has a 10 minutes time limit: for longer runs please use the "batch" mode.

A parallel program can be executed interactively only by submitting an "Interactive" SLURM batch job, using the "srun" command: the job is queued and scheduled as any other job, but when executed, the standard input, output, and error streams are connected to the terminal session from which srun was launched.

For example, to start an interactive session with the MPI program "myprogram", using one node and two processors, you can launch the command:

```
> salloc -N 1 --ntasks-per-node=2 -A <account_name>
```

SLURM will then schedule your job to start, and your shell will be unresponsive until free resources are allocated for you. If not specified, the default time limit for this kind of jobs is one hour.

When the shell returns a prompt inside the compute node, you can execute your program by typing:

```
> srun ./myprogram
```

(srun is recommended with respect to mpirun for this environment)

SLURM automatically exports the environment variables you defined in the source shell, so that if you need to run your program "myprogram" in a controlled environment (i.e. with specific library paths or options), you can prepare the environment in the login shell and be sure to find it again in the interactive shell o the compute node.

On systems using SLURM, you can submit a script *script.x* using the command:

```
> sbatch script.x

You can get a list of defined partitions with the command:

> sinfo
```

For more information and **examples of job scripts**, see section **Batch Scheduler SLURM**.

Submitting serial Batch Jobs

The partition will be available in the full production.

Graphic session

If a graphic session is desired we recommend to use the tool "RCM". Please install the latest version of RCM. See the corresponding paragraph to know more about how to download and use RCM.

Submitting parallel Batch Jobs

To run parallel batch jobs on GALILEO100 you need to specify the partition and the qos that are described in this user guide.

If you do not specify the partition, your jobs will try to run on the default partition g100_all_serial.

The minimum number of cores you can request for a batch job is 1. The **maximum number of cores** that you can request is 3072 (64 nodes). It is also possible to request a **maximum walltime of 24 hours**. Defaults are as follows:

- If you do not specify the walltime (by means of the #SBATCH --time directive), a default value of **30 minutes** will be assumed.
- If you do not specify the number of cores (by means of the "SBATCH -n" directive) a default value of **1 core** will be assumed.
- If you do not specify the amount of memory (as the value of the "SBATCH --mem" DIRECTIVE), a default value of **7800 MB per core** will be assumed on g100_usr_prod and g100_usr_smem partitions. While on the g100_usr_bmem partition, the default value of memory assumed is **632 00 MB per core**.

The maximum memory per node is **375300MB (366.5GB) for thin and viz nodes, about 3TB for fat nodes.** A study on the performances of the thin and fat nodes is ongoing.

On GALILEO100 there are four partitions dedicated to the production:

- *g100_usr_prod*: this partition collects all the thin and persistent memory nodes (memory per node **375300 MB**).
- *g100_usr_smem*: this partition is composed only of thin nodes (memory per node **375300 MB**).
- *g100_usr_pmem*: this partition is composed only of Persistent memory nodes - more info in a following update (memory per node **375300 MB**).
- *g100_usr_bmem*: this partition is composed only of fat nodes (memory per node **3 TB**). It is reserved for jobs that need more than 375300 MB of memory per node and is out of the g100_usr_prod partition.

We encourage you to use the partition more suitable for your job. For more information on the partitions look to Summary.

Processor affinity

Processor affinity, or CPU pinning, enables the binding of processes and threads to a CPU (or group of CPUs). It is crucial to ensure the correct affinity so to avoid the CPUs overallocation, with a significant reduction of performances. It becomes a critical matter when you ask for a full node but, for your specific reasons (memory needs etc.) you don't use all the cores.

The following indications work when running your executables with **srun**, which is the recommended option against mpirun. We refer to a hybrid MPI /OpenMP case compiled with the Intel oneAPI suite.

Given your optimal value of OMP_NUM_THREADS and number of processes, to obtain the full node ask for a number of task such that (--ntasks-per-node * --cpus-per-task)= 48.

- To avoid the processes over allocation of cores rely on the **--cpu-bind=cores** option of srun (you can skip it if you use all the requested cores)
- To enforce the threads affinity use the Intel parameter KMP_AFFINITY, or the OpenMP parameter OMP_PLACES
- To distribute the MPI tasks consecutively inside the sockets, use the -m block:block option of srun (or the equivalent sbatch directive #SBATCH -m block:block)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=12
#SBATCH --cpus-per-task=4
#SBATCH --account=<your_account>
```

```
module load autoload intelmpi/oneapi-2021--binary

export OMP_NUM_THREADS=4

export KMP_AFFINITY=compact    # or OMP_PLACES=cores

srun --cpu-bind=cores -m block:block <your_exe>
```

Use of GPUs on Galileo100

to be soon defined

Users with reserved resources

Users of projects that require reserved resources (such as industrial users or users associated to an agreement that involves dedicated resources) will be associated to a QOS qos_ind.

Using the qos_ind (i.e. specifying the QOS in the submission script) , and specifying the partition g100_spc_prod, users associated to the allowed project will run their jobs on reserved nodes in the g100_spc_prod partition with the features and limits imposed for the particular account.

```
>#SBATCH --partition=g100_spc_prod

>#SBATCH --qos=qos_ind
```

Summary

In the following table, you can find all the main features and limits imposed on the SLURM partitions and QOS. For all partitions, if the memory is not explicitly requested, a DefMemPerCPU = 7800 will be assigned for each requested core. If the walltime is not explicitly requested, a DefaultTime=00:30:00 will be assigned to the job. If the partition is not explicitly requested, the default g100_all_serial partition will be assigned.

SLURM partition	QOS	# cores per job	max walltime	max jobs/resources per user	max memory per node (MB)	priority	notes
g100_all_serial (DEFAULT)	*noQOS*	max = 4	04:00:00	4 cores 120 submitted jobs	31,200 (30 GB)	40	on two login nodes **budget-free**
	qos_install	max = 16	04:00:00	16 cores 1 running job	100 GB	40	request to superc@cineca.it
g100_usr_dbg	*noQOS*	max = 2 nodes	01:00:00		375,300 (366 GB)	40	
	qos_ind	Depending on the specific agreement				90	Partition dedicated to specific kinds of users.
g100_usr_prod g100_usr_smem g100_usr_pmem	*noQOS*	min = 1 max = 32 nodes	24:00:00	100 running jobs 120 submitted jobs	375,300 (366 GB)	40	runs on thin and persistent memory nodes
	g100_qos_bprod	min = 1537 (33 nodes) max = 3072 (64 nodes)	24:00:00			60	runs only on thin nodes
	g100_qos_lprod	min = 1 max = 2 nodes	4-00:00:00	2 nodes 100 running jobs 120 submitted jobs		40	runs only on persistent memory nodes
	qos_special	> 32 nodes	> 24:00:00			40	request to superc@cineca.it
g100_usr_bmem	*noQOS*	25 nodes	24:00:00	100 running jobs 120 submitted jobs	3,036,000 (3 TB)	40	runs on fat nodes
g100_usr_interactive	*noQOS*	max = 0.5 node	8:00:00		375,300 (366 GB)	40	on nodes with GPUs --gres=gpu:N (N=1)
g100_meteo_prod	*qos_meteo*		24:00:00		375,300 (366 GB)	40	Partition reserved to meteo services, **NOT opened to production.** Runs on thin nodes

Programming environment

The programming environment of the GALILEO cluster consists of a choice of compilers for the main scientific languages (Fortran, C and C++), debuggers to help users in finding bugs and errors in the codes, profilers to help in code optimisation.

Compilers

You can check the complete list of available compilers on LEONARDO with the command

```
> modmap -c compilers
```

The available compilers are:

- Intel OneAPI (native and recommended)

- GNU Compilers Collection (GCC)
- NVIDIA nvhpc (ex PGI)
- CUDA

Intel Compilers

The native, and recommended, compilers on GALILEO100 are the Intel ones, since the architecture is based on Intel processors and therefore using the Intel compilers may result in a significant improvement in performance and stability of your code. On the cluster is installed the new suite Intel OneAPI. Initialize the environment with the module command:

```
> module load intel/<VERSION>
```

The suite contains the new Intel oneAPI nextgen compilers (icx, icpx, ifx) and the classic compilers (icc, icpc, ifort):

	Classic	oneAPI	Notes
C /C++ compilers	icc/icpc	icx/icpx	• ICX is the Intel nextgen compiler based on Clang/LLVM technology plus Intel proprietary optimizations and code generation, • ICX enables OpenMP TARGET offload to Intel GPU targets (irrelevant on Galileo100) • ICX and ICC Classic use different compiler drivers. The ICC Classic drivers are icc, icpc, and icl. The ICX drivers are icx and icpx. Use icx to compile and link C programs, and icpx for C++ programs. • Unlike the icc driver, icx does not use the file extension to determine whether to compile as C or C+. Users must invoke icpx to compile C+ files. In addition to providing a core C++ Compiler, ICX is the base compiler for the Intel® oneAPI Data Parallel C++ Compiler and its new driver, dpcpp. • **Intel still recommends ICC/ICPC for standard C/C++ applications**
Fortran compilers	ifort	ifx	• The Intel® Fortran Compiler (Beta) IFX is s a new compiler based on the Intel® Fortran Compiler Classic (ifort) frontend and runtime libraries using LLVM backend technology. ifx is released as a Beta version for users interested in trying offloading to supported Intel GPUs using OpenMP* TARGET directives which ifort does not support (irrelevant on Galileo100) • **Intel recommends IFORT for standard Fortran applications**

NOTE:

- ICX is a new compiler. It has functional and behavioral differences compared to ICC. You can expect some porting will be needed for existing applications using ICC. According to Intel, the transition from ICC Classic to ICX is smooth and effortless. However, you must port and tune any existing applications from ICC Classic to ICX. Please refer to the official Intel Porting Guide for ICC Users to DPCPP or ICX
- IFORT is a completely new compiler. According to Intel, although considerable effort is being made to make the transition from ifort to ifx as smooth and as effortless as possible, customers can expect that some effort may be required to tune their application. IFORT will remain Intel's recommended production compiler until ifx has performance and features superior to ifort. Please refer to the official Intel Porting Guide for ifort Users to ifx
- Please refer to the official Intel C++ Developer Guide and Reference and Fortran Developer Guide and Reference for an exhaustive list of compiler options

The documentation can be obtained with the **man** command after loading the relevant module:

```
> man ifort
> man icc
```

Some miscellaneous flags are described in the following:

```
-extend-source    Extend over the 77 column F77's limit
-free / -fixed    Free/Fixed form for Fortran
-ip               Enables additional interprocedural optimization for single-file compilation
-ipo              Enables interprocedural optimization between files - whole program optimisation
-qopenmp          enables the parallelizer to generate multi-threaded code based on OpenMP directives
```

NOTE for the migration from Galileo to Galileo100: In principle, binaries generated on Galileo should work, but we strongly recommend you to reinstall all your software applications since on Galileo100 there is a different Operating System (Centos 8.3).

GNU compilers

The gnu compilers are always available but they are not the best optimizing compilers, especially for an Intel OneAPI-based cluster like GALILEO100.

For a more recent version of the compiler, initialize the environment with the module command:

> module load gnu

The name of the GNU compilers are:

- **g77**: Fortran77 compiler
- **gfortran**: Fortran compiler with "gnu" standard

- **gcc**: C compiler
- **g++**: C++ compiler

The "gnu" standard is the default value for the -std option. It specifies a superset of the latest Fortran standard that includes all of the extensions supported by GNU Fortran, although warnings will be given for obsolete extensions not recommended for use in new code. To change the standard to which the program is expected to conform, set the -std option to one of the possible values (f95, f2003, f2008, f2018, gnu, or legacy).

The documentation can be obtained with the **man** command:

```
> man gfortan
> man gcc
```

Some miscellaneous flags are described in the following:

```
-ffixed-line-length-132       To extend over the 77 column F77's limit
-ffree-form / -ffixed-form    Free/Fixed form for Fortran
-fopenmp                      Enable handling of OpenMP directives "#pragma omp" in C/C++ and "!$omp" in Fortran.
                              When -fopenmp is specified, the compiler generates parallel code according to the
                              OpenMP Application Program Interface v4.5. This option implies -pthread and -
fopenmp-simd
```

Debuggers and Profilers

If your code aborts at runtime, there may be a problem with it. In order to solve it, you can decide to analyze the core file or to run your code using a debugger.

Compiler flags

In both cases, you need to **enable compiler runtime checks**, by putting specific flags during the compilation phase. In the following we describe those flags for the different Fortran compilers: if you are using the C or C++ compiler, please keep in mind that the flags may differ.

The following flags are generally available for all compilers and are mandatory for an easier debugging session:

```
-O0     Lower level of optimisation
-g      Produce debugging information
```

Other flags are compiler-specific and are described in the following.

INTEL Fortran compiler

The following flags are useful (in addition to "-O0 -g") for debugging your code:

```
-traceback                  generate extra information to provide source file traceback at run time
-fp-stack-check             generate extra code to ensure that the floating-point stack is in the expected state
-fvar-tracking              Generates enhanced debug information useful in finding scalar local variables
-fvar-tracking-assignments Generates enhanced debug information useful for breakpoints and stepping. It
 tells the debugger to stop only at machine instructions that achieve the final
 effect of a source statement.
-check bounds               enables checking for array subscript expressions
-fpe0                       allows some control over floating-point exception handling at run-time
```

Please also refer to the debug C++ compiler option and the debug Fortran compiler option.

GNU Fortran compilers

The following flags are useful (in addition to "-O0 -g") for debugging your code:

```
-Wall            Enables warnings pertaining to usage that should be avoided
-fbounds-check   Checks for array subscripts
```

In the following we report information about some ways to debug your codes:

GNU: gdb (serial debugger)

GDB is the GNU Project debugger and allows you to see what is going on 'inside' your program while it executes -- or what the program was doing at the moment it crashed.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

More details in the online documentation, using the "man gdb" command.

To use this debugger, you should compile your code with one of the gnu compilers and the debugging command-line options described above, then you run your executable inside the "gdb" environment:

```
> module load gnu
> gfortran -O0 -g -Wall -fbounds-check -o myexec myprog.f90
> gdb ./myexec
```

Core file analysis

In order to understand what problem was affecting your code, you can also try a "Core file" analysis. Since core files are usually quite large, be sure to work in the scratch area.

There are several steps to follow:

1. Increase the limit for possible core dumping

```
> ulimit -c unlimited (bash)
> limit coredumpsize unlimited (csh/tcsh)
```

1. If you are using Intel Fortran compilers, set to TRUE the **for_dump_core_file** (or **decfort_dump_flag**) environment variable

```
> export for_dump_core_file=TRUE  (bash)
> setenv for_dump_core_file TRUE  (csh/tcsh)
```

1. Compile your code with the debug flags described above.
2. Run your code and create the core file.
3. Analyze the core file using different tools depending on the original compiler.

INTEL compilers

```
> module load intel
> ifort -O0 -g -traceback -fp-stack-check -check bounds -fpe0 -o myexec prog.f90
> ulimit -c unlimited
> export for_dump_core_file=TRUE
> ./myexec
> ls -lrt
  -rwxr-xr-x 1 aer0 cineca-staff   9652 Apr  6 14:34 myexec
  -rw------- 1 aer0 cineca-staff 319488 Apr  6 14:35 core.25629
```

GNU Compilers

```
> module load gnu
> gfortran -O0 -g -Wall -fbounds-check -o myexec prog.f90
> ulimit -c unlimited
> ./myexec
> ls -lrt
  -rwxr-xr-x 1 aer0 cineca-staff   9652 Apr  6 14:34 myexec
  -rw------- 1 aer0 cineca-staff 319488 Apr  6 14:35 core.25555
> gdb ./myexec core.2555
```

VALGRIND - will be soon available

Valgrind is a framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail. The Valgrind distribution currently includes six production-quality tools: a memory error detector, two thread error detectors, a cache and branch-prediction profiler, a call-graph generating cache profiler, and a heap profiler.

Valgrind is Open Source / Free Software, and is freely available under the GNU General Public License

Totalview
Totalview is a parallel debugger with a practical GUI that assist users to debug their parallel code. It has functionalities like stopping and reprising a code mid-run, setting breakpoints, checking the value of variables anytime, browse between the different tasks and threads to see the different behaviours, memory check functions and so on. For information about how to run the debugger (by connecting the compute nodes to your display via RCM), type the command:

```
> module help totalview
```

Scalasca
Scalasca is a tool for profiling parallel scientific and engineering applications that make use of MPI and OpenMP.
Details how to use scalasca in
http://www.scalasca.org/software/scalasca-2.x/documentation.html

Profilers

In software engineering, **profiling** is the investigation of a program's behaviour using information gathered as the program executes. The usual purpose of this analysis is to determine which sections of a program to optimize - to increase its overall speed, decrease its memory requirement or sometimes both.

A (code) *profiler* is a performance analysis tool that, most commonly, measures only the frequency and duration of function calls, but there are other specific types of profilers (e.g. memory profilers) in addition to more comprehensive profilers, capable of gathering extensive performance data.

Intel VTUNE profiler

Intel performance analisys toolkit that can be used to identify bottlenecks in an application. The tool can be used to perform different types of analisys.

Important Note: for the hardware event-based sampling (EBS), the Intel VTune Profiler profiles your application using the counter overflow feature of the Performance Monitoring Unit (PMU), and the vtune sampling drivers need to be loaded. Ask superc@cineca.it to be authorized to access an environment with the drivers requesting the association of the qos_vtune QOS.

Once enabled to the use of the qos_vtune, you need to request it in your job together with the vtune constraint as follows:

#SBATCH --qos=qos_vtune (or -q qos_vtune)

#SBATCH --constraint=vtune (or -C vtune)

To start the analisys use the command line interface:

```
module load autoload vtune
amplxe-cl -collect hotspots -r <application_path> <vtune_options> <application>
```

To explore the results of the analisys performed use the Intel VTune Amplifier GUI (amplxe-gui or vtune-gui):

```
amplxe-gui <application_path>
```

Please note that to get a correct result in terms of CPU time it is suggested to use of the advanced-hotspots analisys. For example:

```
amplxe-cl -collect advanced-hotspots --target-duration-type veryshort <executable>
```

IMPORTANT: vtune GUIs needs a consistent amount of memory, hence they may fail with a bus error if launched on the login nodes. Furthermore, they benefit GPUs (to be launched with "vglrun vtune-gui"). Please, use the RCM service, either in the SSH mode (on the GPU-equipped node) or SLURM mode.

Scientific libraries (MKL)

MKL

The Intel Math Kernel Library (Intel MKL) enables improving performance of scientific, engineering, and financial software that solves large computational problems. Intel MKL provides a set of linear algebra routines, fast Fourier transforms, as well as vectorized math and random number generation functions, all optimized for the latest Intel processors, including processors with multiple cores.

Intel MKL is thread-safe and extensively threaded using the OpenMP technology.

Examples can be found by loading the mkl module and searching in the directory:

```
${MKLROOT}/examples
```

To use the MKL in your code you to load the module, then to define includes and libraries at compile and linking time, please refer to the Intel oneAPI Math Kernel Library Link Line Advisor.

Parallel programming

The parallel programming on GALILEO100 is based on IntelMPI and OpenMPI versions of MPI. The libraries and special wrappers to compile and link the personal programs are contained in several modules, one for each supported suite of compilers.

These command names refer to wrappers around the actual compilers, they behave differently depending on the module you have loaded.

IntelMPI

IntelMPI in GALILEO100 is recommended since the architecture is based on Intel processors.
To load IntelMPI on GALILEO100, check the module versions available with the "module avail" command, then load the relevant module. After you have loaded the Intelmpi module

```
> mpiifort -o myexec myprof.f90 (uses the ifort compiler)
```

For more option of the compiler, please see

```
> man mpiifort
```

The three main parallel-MPI commands for compilation with OpenMPI are:

- mpiifort (Fortran90/77)
- mpiicc (C)

- mpiicpc (C++)

OpenMPI
On GALILEO100, "gnu" versions of OpenMPI are available. After you have load the openmpi module (check the available version with the "module avail" command):

```
> mpif90 -o myexec myprof.f90 (uses the gfortran compiler)
```

The four main parallel-MPI commands for compilation with OpenMPI are:

- mpif90 (Fortran90)
- mpif77 (Fortran77)
- mpicc (C)
- mpiCC (C++)

In all cases the parallel applications have to be executed with the recommended command:

```
> srun ./myexec
```

There are limitations to running parallel programs in the login shell. You should use the "Interactive SLURM" mode, as described in the "Interactive" section, previously on this page.
