Contents

Curie's Manual

If you have suggestions or remarks, please contact us : hotline.tgcc@cea.fr

Curie's Configuration

Curie is composed of three different architecture :

- Curie fat nodes :
  - Curie fat consists in 360 nodes which contains 4 eight cores CPU Nehalem-EX clocked at 2.27 GHz, let 32 cores / node and 11520 cores for the full fat configuration
  - Each node has 128 Go of memory, let 4 Go / core by default
- Curie thin nodes (not available) :
  - Curie thin consists in 5040 nodes which contains 2 eight cores CPU Sandy Bridge clocked at 2.3 GHz (AVX), let 16 cores / node and 80640 cores for the full thin configuration
  - Each node has 64 Go of memory, let 4 Go / core by default
- Curie hybrid nodes :
  - Curie hybrid consists in 144 nodes which contains 2 GPU Nvidia M2090 coupled to 2 four cores CPU Westmere-EP clocked at 2.67 GHz, let 8 cores and 2 GPU / node and 1152 cores and 288 GPU for the full hybrid configuration
  - Each node has 24 Go of memory, let 3 Go / core by default, and each GPU has 6 Go

System Access

How to reach the system

From your local machine, you need to use the *ssh* command to access curie. *ssh* is a program for logging into a remote machine and for executing commands on it.

```
-bash-4.1$ ssh login@curie.ccc.cea.fr
 password: ****
```

If you need a graphical environment you have to use the -X option :

```
-bash-4.1$ ssh -X login@curie.ccc.cea.fr
```

To log out from Curie, you can use the *Ctrl-d* command, or *exit*

If you have problems for authentifying, you can try *-Y* option.

File transfer

To transfert files between Curie and your local machine, you can use the *scp* command.

Create an archive with the directories you want to copy (it will be faster to transfer) :

```
-bash-4.1$ tar -cvzf archivename.tgz directoryname 1 directoryname 2
```

or in case of a file :

```
-bash-4.1$ tar -cvzf archivename.tgz filename
```

Transfer the archive to Curie :

```
-bash-4.1$ scp archivename.tgz login@curie.ccc.cea.fr:/ccc/cont***/home/login
```

Uncompress the archive in your target directory :

```
-bash-4.1$ tar -xvzf archivename.tgz destinationdirectory
```

Available File Systems

Four file systems are available :

- HOME :
    - I/O perf: slow (NFS)
    - Quota: 3GB per user
    - Use: sources , job submission scripts, parameter files...
    - Commentary: Data are saved
    - Reachable from all resources of the center
    - Environment variable: **$HOME**

- SCRATCH :
    - I/O perf: fastest (Lustre)
    - Quota: 20 TB and 2 000 000 files or directories per user
    - Use: Data, Code output,...
    - Commentary: SCRATCH can be purged if the global free space is too small. However, a minimum lifetime is guaranteed (except hardware failure).
    - Local to Curie
    - Environment variable: **$SCRATCHDIR**
    - Bandwidth : 150 GB/s

- WORK :
    - I/O perf: fast (Lustre via routers)
    - Quota: 1 TB and 500 000 files or directories per user
    - Use: commonly used file (Source code, Binary,...)
    - Commentary: WORK's size is smaller than SCRATCH, it's only managed through quota. There is no purge and no save.
    - Reachable from all resources of the center
    - Environment variable: **$CCCWORKDIR**
    - Bandwidth : 100 GB/s

- STORE :
    - I/O perf: fast (Lustre via routers + HPSS + Tape)
    - Quota: 100 000 files or directories per user
    - Use: data archiving for large files (direct computation allowed in that case) or packed data (tar files, ...)
    - Important:
        - Expected file size range 1Go-100Go
        - Backup mechanism relies on file modification time: avoid using cp options like -p, -a,

    - Reachable from all resources of the center
    - Environment variable: **$CCCSTOREDIR**
    - Bandwidth : 100 GB/s

Inappropriate usage might stop the production

ccc_quota gives information about your current usage of the filesystems:

```
bash-4.0$ $ ccc_quota
Disk quotas for user xxxxxx (uid xxxxx):
             ------------------ VOLUME --------------------  ------------------- INODE --------------------
Filesystem       usage        soft        hard       grace       files       soft        hard       grace
----------       -----        ----        ----       -----       -----       ----        ----       -----
      home          3G          3G          3G           -           -           -           -           -
      work     903.68G        1.0T        1.1T           -       5.07K      500.0K      501.0K           -
     store           4        4.0T        4.1T           -           1      100.0K      101.0K           -
```

You have the size (VOLUME) and the number of files or directories (INODES).

Environment

Operating System

Operating system on Curie's nodes is Bullx Supercomputer Suite AE2.2, based on Red Hat Enterprise Linux 6.

Available shells

The default shell is bash. ksh, csh, tcsh and zsh are also available. We strongly recommand you to use bash shell (Only bash and csh are supported by the support team).

Passwords

You will often need to change your password. This can be done thanks to the *kpasswd* command :

```
-bash-4.1$ kpasswd
 Changing password for user **.
```

Restore lost files

Contact hotline.tgcc@cea.fr or +33 177574242

Text editors

- vi
- emacs
- nano
- nedit
- gedit

"module" command

module allows to change easily the shell environment by initializing, modifying or unsetting environment variables. This option gives you a complete environment to launch a software or to link your code with a library.

The command line option *list* indicates the loaded modules in your environment:

```
-bash-4.1$ module  list Currently Loaded Modulefiles:   1) intel/12.0.084(default)   2) bullmpi/0.18.1(default)
```

The command line option *avail* gives all the available modules :

```
-bash-4.1$ module avail
------------------------------ /usr/local/ccc_users_env/modules/softwares -----------------------------------
abinit/6.4.2    cpmd/3.13.2     espresso/4.2.1  gaussian/09-B01 gromacs/4.5.3   namd/2.7        saturne/2.0.0   siesta/2.0.2    vasp/5.2.11
------------------------------ /usr/local/ccc_users_env/modules/development ---------------------------------
cmake/2.8.3    ddd/3.3.12     jdk/1.6.0_23   papi/4.1.1     paraver/3.99   scalasca/1.3.2 swig/2.0.1     valgrind/3.6.0
------------------------------ /usr/local/ccc_users_env/modules/mpi -----------------------------------------
bullmpi/0.17.2          bullmpi/0.18.1(default)
------------------------------ /usr/local/ccc_users_env/modules/libraries -----------------------------------
boost/1.45.0    fftw3/3.2.2     hdf5/1.8.5      mumps/4.9.2     parmetis/3.1.1  phdf5/1.8.5     qt/4.7.1 fftw2/2.1.5     gsl/1.14        metis/4.0.1     netcdf/4.1.1    petsc/3.1       ptscotch/5.1.11 scotch/5.1.11
------------------------------ /usr/local/ccc_users_env/modules/compilers -----------------------------------
gcc/4.5.1               intel/12.0.084(default)
```

The command line options *load* and *unload* respectively enable to load and unload a module:

```
-bash-4.1$ module  list
Currently Loaded Modulefiles:
1) intel/12.0.084(default)   2) bullmpi/0.18.1(default)
-bash-4.1$ module  unload bullmpi/0.18.1
-bash-4.1$ module  list
Currently Loaded Modulefiles:
1) intel/12.0.084(default)
-bash-4.1$ module  load bullmpi/0.17.2
-bash-4.1$ module  list
Currently Loaded Modulefiles:
1) intel/12.0.084(default)   2) bullmpi/0.17.2
```

The command line option *switch* does the previous operation in one command line:

```
-bash-4.1$ module  switch bullmpi bullmpi/0.17.2
-bash-4.1$ module  list
Currently Loaded Modulefiles:
1) intel/12.0.084(default)   2) bullmpi/0.17.2
```

The command line option *show* indicates how the environment is changed by loading a module. The option *help* gives information about the specified module.

```
-bash-4.1$ module  help gcc/4.5.1
----------- Module Specific Help for 'gcc/4.5.1' ------------------
Name        : gcc
Description : GNU C, C++ and Fortran compilers
Version     : 4.5.1
Web Site    : http://gcc.gnu.org/
-bash-4.1$ module  show gcc/4.5.1
-------------------------------------------------------------------
/usr/local/ccc_users_env/modules/compilers/gcc/4.5.1:

module-whatis    GNU Compiler Collection
conflict         gcc
prepend-path     PATH /usr/local/gcc-4.5.1/bin
prepend-path     LIBRARY_PATH /usr/local/gcc-4.5.1/lib
prepend-path     LD_LIBRARY_PATH /usr/local/gcc-4.5.1/lib:/usr/local/gcc-4.5.1/lib64
prepend-path     MANPATH /usr/local/gcc-4.5.1/man
prepend-path     INFOPATH /usr/local/gcc-4.5.1/info
prepend-path     CPATH /usr/local/gcc-4.5.1/include
prepend-path     FPATH /usr/local/gcc-4.5.1/include
-------------------------------------------------------------------
```

Advice : in most of modules, we set some environment variables like $MKL_LIBS or $FFTW3_INC_DIR which point to library or path. We strongly recommand you to use them in your Makefile. For example when you switch between newer modules, theses variables will be there (but they will point to another library or path).

Compiling / Basis Porting

Available compilers

The available compilers on the cluster are:

- Intel Compiler suite (icc, icpc, ifort)
- GNU compiler suite (gcc, g++, gfortran)
- PGI compiler suite (pgcc, pgCC, pgf90)

To know which version is installed, use the command

```
bash-4.1$ module avail
```

We strongly recommend you to use the Intel Compiler Suite which provides the best performances.

Compiler flags

C/C++

Intel compilers: icc and icpc. Compilation options are the same, except for the the C language behavior. icpc manages all the source files as C++ files whereas icc makes a difference between both of them.

- Basic flags :
    - -o *exe_file* : names the executable exe_file
    - -c : generates the correspondant object file. Does not create an executable.
    - -g : compiles in a debugging mode - R.E. 'Debugging'.
    - -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

- Optimizations :
    - -O0, -O1, -O2, -O3 : optimisation levels - default : -O2

- 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=<value> : defines the "name" variable
    - -M : creates a list of dependance

- Practical :
    - -p : profiling with gprof (needed at the compilation)
    - -mp, -mp1 : IEEE arithmetic, mpl is a compromise between time and accuracy

Fortran

Intel compiler : ifort (Fortran compiler).

- Basic flags :
    - -o *exe_file* : names the executable exe_file
    - -c : generates the correspondant object file does not create an executable.
    - -g : compiles in debugging mode - R.E. 'Debugging'
    - -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

- Optimizations
    - -O0, -O1, -O2, -O3 : optimization levels - default : -O2

- Run-time check
    - -C or -check : generates a code which ends up in 'run time error' (ex : segmentation fault)

- 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=<value> : defines the "name" variable
    - -M : creates a list of dependances
    - -fpp : preprocess the files and compiles

- Practical :
    - -p : profiling with gprof (needed at the compilation)
    - -mp, -mp1 : IEEE arithmetic, mpl is a compromise between time and accuracy
    - -i8 : promotes integers on 64 bytes by default
    - -r8 : promotes reals on 64 bytes by default
    - -module <dir> : send/read the files *.mod in the dir directory

- -fp-model strict : Tells the compiler to strictly adhere to value-safe optimizations when implementing floating-point calculations and enables floating-point exception semantics. It might slow down your program.

Should you wish further information, please refer to the 'man pages' of the compilers.

GNU

- Debugging :
  - -Wall: Short for “warn about all,” this flag tells gfortran to generate warnings about many common sources of bugs, such as having a subroutine or function with the same name as a built-in one, or passing the same variable as an intent(in) and an intent(out) argument of the same subroutine.
  - -Wextra: In conjunction with -Wall, gives warnings about even more potential problems. In particular, -Wextra warns about subroutine arguments that are never used, which is almost always a bug.
  - -w : Inhibits all warning messages (Not adviced)
  - -Werror : Makes all warnings into errors.

Available numerical libraries

MKL Library

Intel MKL library is integrated in the Intel package and contains :

- BLAS, SparseBLAS;
- LAPACK, ScaLAPACK;
- Sparse Solver, CBLAS ;
- Discrete Fourier and Fast Fourier transform (contains the FFTW interface, R.E. FFTW).

If you don't need ScaLAPACK :

```
ifort -o myexe myobject.o ${MKL_LIBS}
```

If you need ScaLAPACK :

```
mpif90 -o myexe myobject.o ${MKL_SCA_LIBS}
```

We provide multithreaded versions for compiling with MKL:

```
ifort -o myexe myobject.o ${MKL_LIBS_MT}
mpif90 -o myexe myobject.o ${MKL_SCA_LIBS_MT}
```

To use multithreaded MKL, you have to set the OpenMP environment variable OMP_NUM_THREADS.

We strongly recommand you to use those variables.

Other libraries

Please see the other softwares section

Parallel Programming

MPI

Available MPI Implementations

Bullxmpi

The default MPI implementation is Bullxmpi, a library provided by Bull. It is based on OpenMPI.

```
curie 50$ module list
Currently Loaded Modulefiles:
 1) oscar-modules/1.0.3       3) fortran/intel/12.0.3.174    5)intel/12.0.3.174(default)
 2) c/intel/12.0.3.174        4) mkl/12.0.3.174              6)bullxmpi/1.1.8.1(default)
curie 50$ ompi info -a
```

The default version of Bullxmpi is given by the command *module list*.

Compiling MPI program

MPI runs using mpicc, mpic++, mpif77 and mpif90 wrappers for compiling and linking MPI programs.

```
curie 50$ mpicc -c test.c
curie 50$ mpicc -o test.exe test.o
```

By default, those 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 :

```
curie 50$ module load gcc
curie 50$ module list
Currently Loaded Modulefiles:
 1) oscar-modules/1.0.3         2) c/intel/12.0.3.174          3) fortran/intel/12.0.3.174     4) mkl/12.0.3.174               5) intel/12.0.3.174(default)    6) bullxmpi/1.1.10.1(default)   7) gcc/4.5.1
curie 50$ mpicc -show
icc -I/opt/mpi/bullxmpi/1.1.8.1/include -pthread -L/opt/mpi/bullxmpi/1.1.8.1/lib -lmpi -ldl -Wl,--export-dynamic -lnsl -lutil -lm -ldl
curie 50$ export OMPI_CC=gcc
curie 50$ mpicc -show
gcc -I/opt/mpi/bullxmpi/1.1.8.1/include -pthread -L/opt/mpi/bullxmpi/1.1.8.1/lib -lmpi -ldl -Wl,--export-dynamic -lnsl -lutil -lm -ldl
```

The *-show* option includes all the libraries and header needed to use MPI.

OpenMP

The Intel and GNU compilers support OpenMP. Intel compilers flags : *-openmp*

```
-bash-4.1$ ifort -openmp -o prog.exe prog.f90
```

GNU compilers flags : *-fopenmp*

```
-bash-4.1$ gcc -fopenmp -o prog.exe prog.c
```

GPU

CUDA

CUDA compiler is available on Curie/hybrid to compile GPU-accelerated programs.

```
curie 50$ module load cuda
curie 50$ module li
Currently Loaded Modulefiles:
 1) c/intel/12.0.4.191          2) fortran/intel/12.0.4.191    3) mkl/12.0.4.191              4) intel/12.0.4.191            5) bullxmpi/1.1.10.1(default)   6) cuda/4.0
```

To compile a simple CUDA code:

```
curie 50$ nvcc -arch=sm 20 -o prog.exe prog.cu
```

To compile a hybrid CUDA code:

```
curie 50$ ls
   cuda.cu    prog.c
curie 50$ module load cuda
curie 50$ icc -c prog.c
curie 50$ nvcc -arch=sm_20 --ccbin=icc -c cuda.cu
curie 50$ icc -o prog_cuda.exe -L$(CUDA_ROOT)/lib64 -lcudart
```

The CUDA module sets environments variables (like CUDA_ROOT) which gives access to CUDA SDK for example:

```
curie 50$ module show cuda
-------------------------------------------------------------------
/usr/local/ccc_users_env/modules/compilers/cuda/4.0:

module-whatis    NVIDIA Compute Unified Device Architecture
conflict         cuda
prepend-path     PATH /usr/local/cuda-4.0/bin
prepend-path     PATH /usr/local/cuda-4.0/computeprof/bin
prepend-path     LD_LIBRARY_PATH /usr/local/cuda-4.0/lib64
prepend-path     LD_LIBRARY_PATH /usr/local/cuda-4.0/computeprof/bin
setenv           CUDA_LIB_DIR /usr/local/cuda-4.0/lib64
setenv           CUDA_ROOT /usr/local/cuda-4.0
setenv           CUDA_SDK_ROOT /usr/local/cuda-4.0/sdk/C
setenv           NV_OPENCL_SDK_ROOT /usr/local/cuda-4.0/sdk/OpenCL
setenv           NV_OPENCL_INC_DIR /usr/local/cuda-4.0/sdk/OpenCL/common/inc
-------------------------------------------------------------------
```

OpenCL

NVIDIA provides tools to compile OpenCL programs. It will be loaded with CUDA module.

```
curie 50$ module load cuda
curie 50$ gcc -I${NV_OPENCL_INC_DIR} -o prog_ocl.exe prog.c -lOpenCL
```

Job submission

Job submissions, resources allocations and the jobs launching over the cluster are managed by SLURM. Special commands prefixed by *ccc_* are provided to execute these operations. To submit a batch job, you first have to write a shell script which contains:

- a set of directives. These directives are lines beginning with **#MSUB** which describes needed resources for your job.

- how to execute your code.

Then your job can be launched by submitting this script to SLURM. The job will enter into a batch queue. When resources are available, the job will be launched over allocated nodes. Jobs can be monitored.

The following paragraphs describe *ccc_** commands and gives some examples of script for different types of jobs..

"ccc_mprun" command

ccc_mprun allows to launch parallel jobs over nodes allocated by resources manager:

```
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. However, you can precise or change its behavior with the command line options:

- -n *nproc* : number of tasks to run
- -c *ncore* : number of cores per task
- -N *nnode* : number of nodes to use
- -M *mem* : required amount of memory per core in Mo
- -T *time* : maximum walltime of the allocations in seconds
- -x : requests exclusive usage of allocated nodes. This is the default configuration for jobs on more that 128 cores.
- -E *extra* : extra parameters to pass directly to the underlying resource mgr
- -K : only allocates resources. If a program is defined it will be executed only once. It would contain ccc_mprun calls to launch parallel commands using the allocated resources.
- -e ' *options* ' : additional parameters to pass to the mpirun command
- -d *ddt* : launches the application in debug mode using DDT

Type *ccc_mprun -h* for an updated and complete documentation.

Script examples

Sequential job

```
#!/bin/bash
#MSUB -r MyJob              # Request name
#MSUB -n 1                 # Number of tasks to use
#MSUB -T 600                # Elapsed time limit in seconds of the job (default: 1800)
#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 raxxxx              # Project ID
#MSUB -q large               # Choosing large nodes
##MSUB -@ noreply@cea.fr:end # Uncomment this line for being notified at the end of the job by sending a mail at the given address


set -x
cd ${BRIDGE_MSUB_PWD}        # BRIDGE_MSUB_PWD is a environment variable which contains the directory where the script was submitted
ccc mprun ./a.out
```

Parallel MPI job

```
#!/bin/bash
#MSUB -r MyJob_Para              # Request 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 standard
#MSUB -A paxxxx                 # Project ID


set -x
cd ${BRIDGE_MSUB_PWD}
ccc_mprun ./a.out
#or
# ccc_mprun -n 32 ./a.out
#or
```

```
# ccc_mprun -n ${BRIDGE_MSUB_NPROC} ./a.out
# BRIDGE_MSUB_NPROC represents the number of tasks
```

Parallel OpenMP/Multithreaded job

```
#!/bin/bash
#MSUB -r MyJob_Para                # Request 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
#MSUB -q standard                  # Choosing standard nodes
#MSUB -A raxxxx                    # Project ID


set -x
cd ${BRIDGE_MSUB_PWD}
export OMP_NUM_THREADS=16
ccc_mprun ./a.out

#!/bin/bash
#MSUB -r MyJob_Para                # Request 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
#MSUB -q large                    # Choosing large nodes
#MSUB -A raxxxx                    # Project ID


set -x
cd ${BRIDGE_MSUB_PWD}
export OMP_NUM_THREADS=${BRIDGE_MSUB_NCORE} # BRIDGE_MSUB_NCORE represents the number of core dedicated per task
ccc mprun ./a.out
```

Warning : an OpenMP/Multithreaded 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 Multithreaded/MPI

```
#!/bin/bash
#MSUB -r MyJob_ParaHyb              # Request 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 standard                  # Choosing standard nodes
#MSUB -A paxxxx                    # 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.
```

You can ask the number of nodes you need:

```
#!/bin/bash
#MSUB -r MyJob_ParaHyb              # Request name
#MSUB -n 4                       # Total number of tasks to use
#MSUB -c 16                      # Number of threads per task to use
#MSUB -N 4                       # Number of nodes
#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 large                    # Choosing large nodes
#MSUB -A paxxxx                    # Project ID


set -x
cd ${BRIDGE_MSUB_PWD}
export OMP_NUM_THREADS=16
ccc mprun ./a.out # This script will launch 4 MPI tasks over 4 nodes (.ie. one task MPI per node). Each task will have 16 OpenMP threads.
```

GPU job

Simple one GPU job:

```
#!/bin/bash
#MSUB -r GPU_Job                  # Request name
#MSUB -n 1                       # Total 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 hybrid                    # Hybrid partition of GPU nodes
#MSUB -A paxxxx                    # Project ID

set -x
cd ${BRIDGE_MSUB_PWD}
module load cuda
ccc_mprun ./a.out
```

You should use *ccc_mprun* to run your GPU code because *ccc_mprun* manages the binding of processes (see Advanced usage page, section process binding).

Hybrid MPI/GPU job:

```
#!/bin/bash
#MSUB -r MPI_GPU_Job               # Request name
#MSUB -n 8                       # Total number of tasks to use
#MSUB -N 4                       # Total number of nodes 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 hybrid                    # Hybrid partition of GPU nodes
```

```
#MSUB -A paxxxx                  # Project ID

set -x
cd ${BRIDGE_MSUB_PWD}
ccc_mprun ./a.out
```

Curie hybrid nodes have 2 GPUs per node. This script launches 8 MPI processes over 4 nodes. Don't forget to load cuda module before submitting your job.

```
#!/bin/bash
#MSUB -r MPI_GPU_Job             # Request name
#MSUB -n 8                     # Total number of tasks to use
#MSUB -c 4                     # 1 socket is reserved for 1 MPI process
#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 hybrid                # Hybrid partition of GPU nodes
#MSUB -A paxxxx                  # Project ID

set -x
cd ${BRIDGE_MSUB_PWD}
ccc_mprun ./a.out
```

See Advanced usage page, section process binding for more precisions.

MPMD job

A MPMD job (for Multi Program Multi Data) is a parallel job which launch different executables over the processes.

```
#!/bin/bash
#MSUB -r MyJob_Para                # Request name
#MSUB -n 32                    # Total 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 standard                # Choosing standard nodes
#MSUB -A paxxxx                  # Project ID


set -x
cd ${BRIDGE_MSUB_PWD}
cat << END > app.conf
1      ./bin1                # This script will launch the 3 executables
5      ./bin2                # respectively on 1, 5 and 26 cores
26      ./bin3
END
ccc  mprun -f app.conf
```

"ccc_msub" command

The previous script have to be submitted to the resources manager with *ccc_msub* command:

```
bash-4.1$ cat script.sh
#!/bin/bash
#MSUB -r MyJob_Para                # Request 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 large                 # Choosing large nodes
#MSUB -A raxxxx                  # Project ID


set -x
cd ${BRIDGE_MSUB_PWD}
ccc_mprun ./a.out
bash-4.1$ ccc_msub script.sh
Submitted Batch Session 1556
```

Remark: *#MSUB* directive lines are not necessary. If a directive is not specified, a default value will be initialized.

Directive lines can be specified through command line options to *ccc_msub*. In this case, command line parameters take precedence over script directives.

```
bash-4.1$ cat script.sh
#!/bin/bash
#MSUB -r MyJob_Para                # Request name
#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 large                 # Choosing large nodes
#MSUB -A raxxxx                  # Project ID

set -x
cd ${BRIDGE_MSUB_PWD}
ccc_mprun ./a.out
bash-4.1$ ccc_msub -n 32 -T 1800 script.sh
Submitted Batch Session 1557
```

If one of theses command line options like *-n*, *-N*, *-c* or *-x* is given, it cancels all effects of MSUB directives with *-n*, *-N*, *-c* or *-x*.

We recommend to use the MSUB directives rather than the command line options. Here are some other command line options for *ccc_msub*:

- -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)
- -r *reqname* : job name
- -n *nprocs* : number of tasks that will be used in parallel mode (default=1)
- -c *ncores* : number of cores per parallel task to allocate (default=1)
- -N *nnodes* : number of nodes to allocate for parallel usage
- -T *time_limit* : maximum walltime of the batch job in seconds (default=18000)

- -M *mem_limit* : maximum memory amount required per allocated core in Mb
- -x : request for exclusive usage of allocated nodes
- -X : allow enables X11 forwarding (useful for DDT)
- -A *project* : specify the project id
- -E "*extra_parameters...*" : extra parameters to pass directly to the underlying batch system
- -q *partition* : requested type of node
- -Q *qos* : requested QoS
- -S *starttime* : requested start time using format like "HH:MM" or "MM/DD HH:MM"
- -@ *mailopts* : mail options following the pattern mailaddr["begin|end|begin,end"]
- exp: ccc_msub -@ jdoe@foo.com :begin,end will send a mail to jdoe at the begining
- and the end of the job default behavior depends of the underlying batch system

Type *ccc_msub -h* for an updated and complete documentation.

Don't forget to specify your correct project ID with the **-A** option . Otherwise, you may use hours from another project.

Choosing between Curie's three architectures

When you're submitting your job on Curie, you can choose on which of the three architectures available your job is going to run:

- Curie's standard nodes, using the *-q standard* option.
- Curie's large nodes, using the *-q large* option.
- Curie's hybrid nodes, using the *-q hybrid* option.

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

Test QoS

To develop or debug your code, you may submit a job using the test QoS (Quality of Service) which will allow it to be scheduled faster. This QoS is limited to two jobs of 30 minutes and each job is limited to 8 nodes. The cpu time is accounted normally. To do this, simply add *#MSUB -Q test* in your submission script (see below).

```
#!/bin/bash
#MSUB -r MyJob                 # Request name
#MSUB -n 64                    # Number of tasks to use (256 max for test QoS)
#MSUB -T 1800                  # Elapsed time limit in seconds of the job (1800 max with test QoS)
#MSUB -Q test                  # QoS test
#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 standard              # Choosing standard nodes
#MSUB -A raxxxx                # Project ID

ccc_mprun ./a.out
```

Multi Step job

To launch a multi step job like this:

JOB A ==> JOB B ==> JOB C

where JOB B can be launched only if JOB A is finished, then JOB C can be launched if JOB B is finished.

Here are the corresponding scripts:

JOB_A.sh :

```
#!/bin/bash
#MSUB -r JOB_A
#MSUB -n 32
ccc_mprun ./a.out
ccc_msub JOB_B.sh
```

JOB_B.sh :

```
#!/bin/bash
#MSUB -r JOB_B
#MSUB -n 16
ccc_mprun ./b.out
ccc_msub JOB_C.sh
```

JOB_C.sh :

```
#!/bin/bash
#MSUB -r JOB_C
#MSUB -n 8
ccc mprun ./c.out
```

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

/!\ Be careful, if the job is killed or has reached his time allocation limit, all the job will be removed and the last "ccc_msub" may not be launched. To avoid this case, you can use the ccc_tremain from libccc_user (described below) or use the "#MSUB -w" directive like that:

```
#!/bin/bash
#MSUB -r JOB_A
#MSUB -n 32
#MSUB -w
ccc_msub JOB_A.sh
ccc_mprun ./a.out
```

The directive "#MSUB -w" creates a dependance between jobs **with the same name**. If you submit two jobs with the same name, the second will run only if the first has finished. In our case of multi-step jobs, you submit the next script before *ccc_mprun* command, but the next will be launched after the current job will be done.

Job monitoring and control

ccc_mpp provides information about jobs on the cluster.

```
bash-3.0$ ccc_mpp
USER     GROUP    BATCHID  NCPU QUEUE      STATE   RLIM   RUN  SUSP   OLD  NAME       NODES
login    s8          3117    36 test       RUN     30.0m  3.4m    -  3.4m job_A      curie[22-23]
login    s8          3119    24 test       PEN     30.0m     -    - 31.0s job_B
```

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

ccc_mpeek gives information about a job during its run.

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

ccc_mdel kills jobs:

```
bash-4.1$ ccc_mpp
USER     GROUP    BATCHID  NCPU QUEUE      STATE   RLIM   RUN  SUSP   OLD  NAME       NODES
login    s8          3117    36 test       RUN     30.0m  3.4m    -  3.4m job_A      curie[22-23]
bash-4.1$ ccc_mdel 3117
```

The command *ccc_myproject* gives information about the accounting of your project:

```
bash-4.1$ ccc_myproject
Accounting for project XXXXXXX on Curie at 2011-04-13
Login                          Time in hours
login01     ...........................75382.44
login02     ...............................0.00
Total       ...........................75382.44
Allocated   .........................2000000.00
Percent Used.............................3.77%
Project deadline  201X-0X-0X
```

You will find:

- consumed compute time per project's member
- total consumed compute time
- project's deadline

The accounting is updated once a day.

libccc_user

We provide a library which allows to get information about job. An interesting functionnality is the subroutine *ccc_tremain* which gives the execution time remaining in seconds before the job ends. For example, it is useful if your code runs more than the duration allocated. Then, you can save restart files for a next job.

- C/C++ :

```
#include "ccc_user.h"
...
double time_remain;
int error;
...
error = ccc_tremain(&time_remain);
if(!error)    printf("Time remaining before job ends: %lf seconds\n", time_remain);
...
```

- Fortran :

```
...
double precision :: time_remain
...
call ccc_tremain(time_remain)
print*, 'Time remaining before job ends: ', time_remain, ' seconds'
...
```

We give here an example to compile a program using libccc_user:

```
$ module load libccc_user
$ icc -o prog.exe prog.c ${CCC LIBCCC USER LDFLAGS }
```

Debugging

Compiler flags

Before debugging, you need to compile your code with theses flags:

- - *-g* : Generates extra debugging information usable by GDB. -g3 includes even more debugging information. This option is available for GNU and INTEL version to compile C/C++ and Fortran programs. - *-O0* : Suppress all optimizations.

GNU

Gnu fortran compiler :gfortran

- -fbacktrace: Specifies that if the program crashes, a backtrace should be produced if possible, showing what functions or subroutines were being called at the time of the error.
- -fbounds-check: Add a check that the array index is within the bounds of the array every time an array element is accessed. This substantially slows down a program using it, but is a very useful way to find bugs related to arrays; without this flag, an illegal array access will produce either a subtle error that might not become apparent until much later in the program, or will cause an immediate segmentation fault with very little information about cause of the error.

Intel fortran compiler : ifort

- -traceback : generate extra information to provide source file traceback at run time
- -check bounds : enables checking for array subscript expressions

Available Debuggers

- Gnu : GDB
- Intel : IDB
- DDT : Parallel debbugger from Allinea

To use gnu and intel debugger, use the command *gdb* or *idb*.

DDT

To use it, you need to load a module. For instance :

```
bash-4.1 $ module load ddt
```

Then use the command *ddt*. In case of parallel codes, in your submission script, you need to replace the line

```
mpirun -n 16 ./a.out
```

by :

```
ddt -start -n 16 ./a.out
```

Example of submission script:

```
bash-4.1$ cat ddt.job
#!/bin/bash
#MSUB -r MyJob Para                # Request 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


set -x
cd ${BRIDGE_MSUB_PWD}
ddt -start -n 32 ./a.out
bash-4.1$ ccc_msub -X ddt.job
```

Note : you must submit with *-X* for ccc_msub, if you want X11 forwarding.

PRACE infrastructure

Curie is part of the PRACE infrastructure and access to the internal PRACE network and relative services are available from Curie login nodes.

Note: PRACE services like GSI-SSH and GridFTP require an authorized X.509 grid certificate. To register your grid certificate in CEA authorization database, please provide the Distinguished Name of your X.509 grid certificate to hotline.tgcc@cea.fr.

Note: in-depth documentation of the PRACE services and their use will be soon provided on the PRACE-RI website. In the mean time we provide you guidelines to perform most useful tasks.

Connect to a remote PRACE supercomputer

- To connect with SSH to a remote PRACE supercomputer with your login information for that system:

```
$ ssh jugene5d.zam.kfa-juelich.de -l <your_fzj_login>
```

- If you have a X.509 grid certificate, registered in the authorization database of the remote site you want to connect to, you can also connect with GSI-SSH to that remote system once your grid credential is enabled on Curie

```
$ gsissh jugene5d.zam.kfa-juelich.de -p 2222
```

Note: Please see your grid Certification Authority documentation to learn how to enable your grid credential on Curie and the remote site documentation to learn how to register your grid certificate at this remote site.

Connect to curie from a remote PRACE supercomputer

- From a remote PRACE supercomputer, you can connect with SSH to Curie login nodes with your Curie login information

```
$ ssh curie-prace.ccc.cea.fr -l <your cea login>
```

- If you have a X.509 grid certificate, registered in CEA authorization database, you can also connect to Curie with GSI-SSH once your grid credential is enabled on remote site:

```
$ gsissh curie-prace.ccc.cea.fr -p 2222
```

Note: please see your Grid Certification Authority documentation to learn how to enable your grid credential on remote site.

Note: to register your grid certificate in CEA authorization database, please provide the Distinguished Name of your X.509 grid certificate to hotline.tgcc@cea.fr.

Transfer data between PRACE supercomputers

To transfer data from/to Curie, you can use SCP with the login information on both local and remote supercomputers:

- From a remote PRACE supercomputer

```
$ scp myfile <your cea login>@curie-prace.ccc.cea.fr:/path/to/copy
```

- From a Curie login node

```
$ scp myfile <your_fzj_login>@jugene5d.zam.kfa-juelich.de:/path/to/copy
```

If you have a X.509 grid certificate, registered in both CEA and remote site authorization databases, you can also transfer data with GridFTP:

- From a remote PRACE supercomputer

```
$ globus-url-copy gsiftp://jugene5d.zam.kfa-juelich.de:2812/path/to/sourcefile gsiftp://garbin-prace.eole.ccc.cea.fr:2812/path/to/destdir/
```

- From a Curie login node

```
$ globus-url-copy gsiftp://garbin-prace.eole.ccc.cea.fr:2812/path/to/sourcefile gsiftp://jugene5d.zam.kfa-juelich.de:2812/path/to/destdir/
```

Note: to register your grid certificate in CEA authorization database, please provide the Distinguished Name of your X.509 grid certificate to hotline.tgcc@cea.fr.