Software

How the software is managed in the Cirrus HTC and HPC clusters

Software Management

The INCD software is managed using a tool called modules. This tool allows the user to load the correct environment (PATH, LD_LIBRARY_PATH, etc) for each specific software package. The main commands are explained in the next sections.

List of available Software

- All software available to INCD users is usable using modules, the presented list is context depended, some groups have customized environment bundles showed only to them:

```
[jpina@hpc7 ~]$ module avail


------------------------------------------------------------------------------------------ /cvmfs/sw.el7/modules/LIP
------------------------------------------------------------------------------------------
clhep/2.3.4.3      delphes/3.4.2pre15 geant/4.10.03      lhapdf/6.1.6       pythia/6.4.26      pythia/8.2.35
root/6.08.00       root/6.16.00
delphes/3.4.1      fastjet/3.3.0      hepmc/2.6.9        madgraph/2.6.0     pythia/6.4.28      root/5.34.36
root/6.10.02       root/6.16.00-RUBEN
delphes/3.4.2pre10 geant/4.10.02.p01  heptoptagger/2.0   madgraph/2.6.1     pythia/8.2.15      root/6.04.02
root/6.14.06       xroot/4.3.0


------------------------------------------------------------------------------------------ /cvmfs/sw.el7/modules/soft
------------------------------------------------------------------------------------------
aster-13.1.0             cmake-3.5.2              gcc63/netcdf-fortran/4.4.4 hdf5-1.8.16
ngspice/30               r-3.2.5                  udocker/1.0.4
blat-36.2                cuda-8.0                 gcc63/ngspice/30           homer-4.8               openmpi/2.1.0           r-
3.5.2                    udocker/1.1.0
boost-1.55               cuda-9.0                 gcc63/openmpi/2.1.0        kallisto-0.43.0         openmpi-
1.10.2           rt                       udocker/1.1.0-devel
bowtie2-2.3.0            DATK                     gcc63/openmpi-1.10.2       macs-1.4.2              openmpi-
2.1.0            sbcl-1.3.4               udocker/1.1.1
clang/7.0.0              fastqc-0.11.5            gcc63/openmpi-2.1.0        matlab/R2018a
parallel/20180622        schism/5.4.0             weblogo-2.8.2
clang/ngspice/30         fftw-3.3.4               gcc63/r-3.4.2              mpich-3.2               plumed-2.2.1
sicer-1.1                wine/4.2
clhep-2.2.0.8            fftw-3.3.5               gcc63/schism/5.4.0         mvapich2-1.8            python/3.7.2
```

star-2.5.2b
clhep-2.3.1.1 gcc-4.8 gcc-6.3 netcdf/4.6.1 python-2.7.11 teste
clhep-2.3.2.2 gcc63/mpich-3.2 gcc-7.3 netcdf2/4.6.1 python-3.5.1
trimmomatic-0.33
cmake/3.11.2 gcc63/netcdf/4.6.1 gromacs-4.6.7 netcdf-fortran/4.4.4 python-
3.5.4 udocker/1.0.2
...
Show / list module information
* Using this option will show the following information:

```
[jpina@hpc7 ~]$ module show gcc63/ngspice/30
-------------------------------------------------------------------
/cvmfs/sw.el7/modules/soft/gcc63/ngspice/30:

module-whatis Sets up NGSpice
system test -d /cvmfs/sw.el7/ar/ix_5400/gcc63/ngspice/30
-------------------------------------------------------------------
```

Summary

software[1]	Location[2]	software	availability	complex software[3]	type of compiler[4]
gromacs-4.6.7	/cvmfs/sw.el7/modules/LIP	Gromacs 4.6.7	usable by everyone	no	OS default (gcc -V)
clang/7.0.0	/cvmfs/sw.el7/modules/soft	AMD compiler	usable by everyone	no	clang 7.0.0
gcc63/r-3.4.2	/cvmfs/sw.el7/modules/soft	R software compiled with gcc 6.3	usable by everyone	yes	gcc 6.3

[1] these are just examples and the full list of available software can be accessed by logging-in into the login machine and run command 'module avail'
[2] Path of the installed software. No action needed from the user
[3] This means if a software is composed by more than one module (compiler + software + modules)
[4] At INCD software can be compiled using several types of compilers (gcc, cland (AMD), Intel etc)

Load Software

- Please select a name using previous command

```
module load gcc63/ngspice/30
```

Loaded software

- How to list your loaded modules

```
[jpina@hpc7 ~]$ module list
Currently Loaded Modulefiles:
  1) gcc-6.3           2) gcc63/ngspice/30
```

- On this example two different software gcc 6.3 plus ngsipe 3.9.

Unload software

```
[jpina@hpc7 ~]$ module list
Currently Loaded Modulefiles:
  1) gcc-6.3           2) gcc63/ngspice/30
[jpina@hpc7 ~]$ module unload gcc63/ngspice/30
[jpina@hpc7 ~]$ module list
Currently Loaded Modulefiles:
  1) gcc-6.3
[jpina@hpc7 ~]$ module unload gcc-6.3
[jpina@hpc7 ~]$ module list
No Modulefiles Currently Loaded.
```

- Do no unload gcc-6.3 or ngspice will stop working because this software was compiled against gcc 6.3.

Help

- List usefull comands for modules software

```
[jpina@hpc7 ~]$ module help

  Modules Release 3.2.10 2012-12-21 (Copyright GNU GPL v2 1991):
```

Usage: module [switches] [subcommand] [subcommand-args]

> **NOTE** only free software is available to **ALL** users at INCD. For non-free software please contact the **INCD support helpdesk**

Software List

List of software centrally available via the modules tool at the INCD Cirrus HPC and HTC clusters as of **August 2022**. Full list changes and to request the installation of additional software contact the INCD support **helpdesk**.

Intel Compilers available

Users can also install software on their own for further information see the section on **User Software Installation**. Execution of user defined software environments (operating system and libraries) using Linux containers in the HPC and HTC clusters with **uDocker** and **Singularity** is also supported.

INCD-Lisbon HPC and HTC cluster (Cirrus-A)

AlmaLinux 8

```
[jpina@cirrus01 ~]$ module avail

---------------------------------------------------------------------------------------------------------- /cvmfs/sw.el8/modules/hpc/main
----------------------------------------------------------------------------------------------------------
   R/4.4.3        (D)    gcc-11.3    gcc-8.5             julia/1.9.4     julia/1.11.4         (D)    matlab-runtime/9.0.1
python/3.7     python/3.11 (D)    spack/0.19.1        udocker/1.3.16
   aoc-4.0.0             gcc-13.2    intel/oneapi/2023    julia/1.10.5    matlab-runtime/R2016a        matlab-
runtime/24.1  (D)    python/3.8     qgis/3.40.5         spack/0.21.1 (D)    udocker/1.3.17 (D)
   bazelisk/1.22.0       gcc-14.1    julia/1.6.7         julia/1.11.0    matlab-runtime/R2024a        opam
python/3.10    qgis/3.44.2 (D)    spark/3.1.1

----------------------------------------------------------------------------------------------------------
/cvmfs/sw.el8/modules/hpc/aoc40
----------------------------------------------------------------------------------------------------------
   aoc40/R/4.2.2          aoc40/gromacs/2023     aoc40/libs/blas/3.11.0     aoc40/libs/gsl/2.7
aoc40/libs/jemalloc/5.3.0     aoc40/libs/libpng/1.6.39     aoc40/libs/openblas/0.3.21     aoc40/openmpi/4.1.4
   aoc40/gamess/2023-R2     aoc40/libs/aocl/4.0     aoc40/libs/fftw/3.3.10     aoc40/libs/hdf5/1.14.0
```

aoc40/libs/lapack/3.11.0 aoc40/libs/nlopt/2.7.1 aoc40/mvapich2/2.3.7

/cvmfs/sw.el8/modules/hpc/gcc85

---

gcc85/R/4.2.2 gcc85/libs/gsl/2.7 gcc85/libs/lapack/3.11.0 gcc85/libs/openblas/0.3.21 gcc85/netcdf-cxx/4.2 gcc85/openfoam-org/11 gcc85/plumed/2.9.0
gcc85/libs/blas/3.11.0 gcc85/libs/hdf5/1.14.0 gcc85/libs/libpng/1.6.39 gcc85/mvapich2/2.3.7 gcc85/netcdf-cxx4/4.3.1 gcc85/openfoam/2306 gcc85/spin/6.5.1
gcc85/libs/fftw/3.3.10 gcc85/libs/jemalloc/5.3.0 gcc85/libs/nlopt/2.7.1 gcc85/netcdf-c/4.9.0 gcc85/netcdf-fortran/4.6.0 gcc85/openmpi/4.1.4

/cvmfs/sw.el8/modules/hpc/gcc11

---

gcc11/R/4.2.2 gcc11/gromacs/2022.5 gcc11/libs/boost/1.80.0 gcc11/libs/jemalloc/5.3.0 gcc11/libs/openblas/0.3.21 gcc11/netcdf-cxx4/4.3.1 gcc11/openfoam-org/10 (D) gcc11/paraview/5.10.1 gcc11/slim/4.2.2
gcc11/cp2k/2023.1 gcc11/gromacs/2023 (D) gcc11/libs/fftw/3.3.10 gcc11/libs/lapack/3.11.0 gcc11/mvapich2/2.3.7 gcc11/netcdf-fortran/4.6.0 gcc11/openfoam/2012 gcc11/pfft/1.0.8 gcc11/slim/5.0 (D)
gcc11/gromacs/2020.4 gcc11/keras/2.10.0 gcc11/libs/gsl/2.7 gcc11/libs/libpng/1.6.39 gcc11/netcdf-c/4.9.0 gcc11/ngspice/37 gcc11/openfoam/2206 (D) gcc11/plumed/2.8.0
gcc11/gromacs/2021.4 gcc11/libs/blas/3.11.0 gcc11/libs/hdf5/1.14.0 gcc11/libs/nlopt/2.7.1 gcc11/netcdf-cxx/4.2 gcc11/openfoam-org/2.4.0 gcc11/openmpi/4.1.4 gcc11/plumed/2.9.0 (D)

/cvmfs/sw.el8/modules/hpc/gcc13

---

gcc13/arpack/3.9.1 gcc13/brams/6.0.0 gcc13/foam-extend/4.1 gcc13/libs/hdf5/1.14.3 gcc13/netcdf-c/4.9.2 gcc13/openfoam/2206 gcc13/openmpi/4.1.6 gcc13/spooles/2.2 gcc13/xbeach/1.24.6057
gcc13/blas/3.12.0 gcc13/calculix/2.20 gcc13/gromacs/2020.4 gcc13/mooring gcc13/netcdf-fortran/4.6.1 gcc13/openfoam/2306 gcc13/petsc/3.22 gcc13/vtk/9.4.1 gcc13/xbeach/1.24.6088M (D)
gcc13/boost/1.86.0 gcc13/eigen3/3.4.0 gcc13/lapack/3.12.0 gcc13/mvapich2/2.3.7 gcc13/openblas/0.3.28 gcc13/openfoam/2406 (D) gcc13/precice/3.1.2 gcc13/xbeach/1.23.5960

--- /cvmfs/sw.el8/modules/hpc/intel

---

gmxMMPBSA/1.6.3 intel/libs/blas/3.11.0 intel/libs/hdf5/1.14.0 intel/libs/lapack/3.11.0

intel/mvapich2/2.3.7 intel/netcdf-cxx4/4.3.1 intel/openfoam-org/11 intel/openmpi/4.1.4
intel/castep/24.1 intel/libs/fftw/3.3.10 intel/libs/hdf5/1.14.3 (D) intel/libs/libpng/1.6.39 intel/netcdf-c/4.9.2 intel/netcdf-fortran/4.6.1 intel/openfoam/2306 intel/openmpi/4.1.6 (D)
intel/gamess/2023-R2 intel/libs/gsl/2.7 intel/libs/jemalloc/5.3.0 intel/libs/nlopt/2.7.1 intel/netcdf-cxx/4.2 intel/nwchem/nwchem-7.2.2 intel/openfoam/2506 (D)

---------- /cvmfs/sw.el8/modules/gpu ----------
cuda/10.2 cuda/11.2 cuda/11.8 cuda/12.1 cuda/12.6 (D) nvhpc-byo-compiler/23.1 nvhpc-nompi/23.1 nvhpc/23.1 tensorflow/2.14.0 tensorflow/2.18.0 tensorflow/2.19.0 (D)

---------- /cvmfs/sw.el8/modules/bio ----------
R/4.2.2 bcftools/1.8 bowtie2/2.5.1 (D) fastqc/0.11.9 grenepipe/0.14.0 igv/2.12.3 orca/6.0.1 raxml/8.2.12 star/2.7.6a trinity/2.14.0
R/4.4.3 beagle/5.4 bwa/0.7.17 fastqc/0.12.1 (D) grenepipe/0.15.0 (D) iqtree2/2.1.2 orca/shared/5.0.4 salmon/1.10.3 star/2.7.10b (D) vcftools/0.1.14
admixture/1.3.0 bismark/0.23.0 chimera/1.18 figtree/0.12.1 hisat2/2.2.1 iqtree2/2.2.2 (D) orca/static/5.0.4 samtools/1.8 stringtie/3.0.0 vcftools/0.1.16 (D)
alphafold/2.3.2 bismark/0.24.1 (D) chromopainter/0.0.4 figtree/1.4.3 (D) hmmer/3.3.2 jellyfish/2.3.1 picard/3.1.1 samtools/1.17 (D) suitesparse/7.8.2 vina-gpu/2.1
alphapulldown/2.0.1 blast-plus/2.12.0 cutadapt/4.4 finestructure/4.1.1 hmmer/3.4 (D) kallisto/0.50.1 plink/1.07 sickle/1.33 transdecoder/5.5.0
autodock-gpu-develop/11.3.0 blast-plus/2.14.1 (D) eigensoft/8.0.0 freebayes/1.3.6 htslib/1.8 mrbayes/3.2.7a plink2/2.00a4.3 sratoolkit/3.0.0 transdecoder/5.7.1 (D)
autodock-vina/1.2.3 bowtie2/2.4.2 evigene/23.7.15 gatk/4.5.0.0 htslib/1.17 (D) openbabel/3.0.0 popoolation2/1.201 stAIcalc trimgalore/0.6.10

---------- /cvmfs/sw.el8/modules/LIP ----------
gcc11/geant/4.10.7.3 gcc11/topas/3.9 gcc85/gdcm/2.8.9 gcc85/geant/4.10.7.3 gcc85/root/6.24.06 gcc85/root/6.28.10 gcc85/root/6.30.04 (D) madgraph/2.9.14

- If software required not listed please ask INCD support

Access and Middleware

Besides conventional login using SSH, the cirrus-A computing resources can be accessed via middleware using the Unified Middleware Distribution through the EGI and IBERGRID distributed computing infrastructures.

INCD-D HPC and HTC cluster (Cirrus-D)

Almalinux 8

```
[jpina@cirrus01 ~]$ module avail


------------------------------------------------------------------------------- /cvmfs/sw.el7/modules/hpc
-------------------------------------------------------------------------------
   DATK                          gcc-6.3                       gcc83/gromacs/2021.2            intel/openfoam/1906
python-2.7.11
   aoc22/libs/openblas/0.3.10    gcc-7.3                       gcc83/iqtree2/2.1.3
intel/openfoam/2012             python-3.5.1
   aoc22/openmpi/4.0.3           gcc-7.4                       gcc83/libs/gsl/2.6              intel/openfoam/2112
(D)    python-3.5.4
   aocc/2.2.0                    gcc-7.5                       gcc83/mvapich2/2.3.5            intel/openmpi/4.0.3
python/3.7.2
   aocl/2.2                      gcc-8.3                       gcc83/nlopt/2.6.2               intel/openmpi/4.1.1    (D)
python/3.9.12                 (D)
   aster-13.1.0                  gcc55/openmpi/4.0.3           gcc83/openmpi/4.0.3
intel/swan/41.31              r-3.2.5
   autodock/4.2.6                gcc63/fftw/3.3.9              gcc83/openmpi/4.1.1    (D)    kallisto-0.43.0
r-3.5.2
   beast/1.10.4                  gcc63/libs/blas/3.9.0         gcc83/prover9/2009-11A
libs/32/jemalloc/5.3.0        r-3.6.3
   blat-36.2                     gcc63/libs/gsl/2.6            git/2.9.5                       libs/blas/3.9.0        r-4.0.2
   boost-1.55                    gcc63/libs/lapack/3.9.0       gromacs-4.6.7                   libs/gsl/2.6           sbcl-
1.3.4
   bowtie2-2.3.0                 gcc63/libs/libpng/1.6.37      hdf4/4.2.15                     libs/jemalloc/5.3.0
sicer-1.1
   clang/7.0.0                   gcc63/libs/openblas/0.3.10    hdf5-1.8.16                     libs/lapack/3.9.0
star-2.5.2b
   clang/ngspice/30              gcc63/mpich-3.2               hdf5/1.12.0                     libs/libpng/1.6.37
tensorflow/2.4.1
```

clang/openmpi/4.0.3 gcc63/mvapich2/2.3.5 homer-4.8
libs/openblas/0.3.10 tensorflow/2.7.0 (D)
cmake/3.5.2 gcc63/netcdf-fortran/4.4.4 hwloc/2.1.0 macs-1.4.2
trimmomatic-0.33
cmake/3.11.2 gcc63/netcdf-fortran/4.5.2 (D) intel/2019
matlab/R2018a udocker/1.1.3
cmake/3.17.3 gcc63/netcdf/4.6.1 intel/2020 matlab/R2018b
udocker/1.1.4
cmake/3.20.3 (D) gcc63/netcdf/4.7.4 (D) intel/gromacs/2021.5 matlab/R2019b
(D) udocker/1.1.7
conn-R2018b gcc63/ngspice/34 intel/hdf4/4.2.15 mpich-3.2
udocker/alphafold/2.1.1
cuda gcc63/openmpi/1.10.7 intel/hdf5/1.12.0 mvapich2/2.3.5
udocker/tensorflow/cpu/2.4.1
cuda-10.2 gcc63/openmpi/2.1.0 intel/libs/libpng/1.6.37 netcdf-fortran/4.5.2
udocker/tensorflow/gpu/2.4.1
cuda-11.2 gcc63/openmpi/4.0.3 intel/libs/openblas/0.3.10 netcdf/4.7.4
view3dscene/3.18.0
elsa/1.0.2 gcc63/openmpi/4.1.1 (D) intel/mvapich2/2.3.5 nlopt/2.6.2 (D)
vim/8.2
fastqc-0.11.5 gcc63/r-3.4.2 intel/netcdf-fortran/4.5.2 openmpi/1.10.7
weblogo-2.8.2
fftw/3.3.4 gcc63/schism/5.4.0 intel/netcdf/4.7.4 openmpi/2.1.0
wine/4.2
fftw/3.3.5 (D) gcc63/xbeach/1.23.5527 intel/oneapi/2021.3
openmpi/4.0.3 ww3/6.07.1
freewrl/4.4.0 gcc74/gromacs/2019.4 intel/oneapi/2022.1 (D) openmpi/4.1.1
(D)
gcc-4.8 gcc74/openmpi/4.0.3 intel/openfoam/5.0 parallel/20180622
gcc-5.5 gcc74/plumed/2.5.3 intel/openfoam/8.0 plumed/2.2.1

User Software Installation

Tips and examples on how to install software locally

Allowed directories for users software installations

There are a few options for local users installation locations as showed on the next table, create appropriate paths bellow the base directory.

Site	Base Directory	Comments
INCD-Lisbon	/home/GROUP/USERNAME	
INCD-Lisbon	/data/GROUP/USERNAME	
INCD-Lisbon	/data/GROUP	*available on request* and a responsible must be appointed
INCD-Lisbon	/exper-sw	legacy location, to be moved whenever possible
INCD-Minho	/home/GROUP/USERNAME	
ISEC-COIMBRA	***none so far***	

- GROUP: User group name
- USERNAME: User login name

> **NOTE** Some applications may have dependencies requiring cluster wide installation of packages, please contact the **INCD support helpdesk** on those cases.

Install Miniconda

Small tutorial on how to install and run miniconda on the HPC clusters.

1. Login into your login machine

2. Download Miniconda into your local home

```
$ wget https://repo.anaconda.com/miniconda/Miniconda2-latest-Linux-x86_64.sh
```

3. Execute Miniconda

```
$ chmod +x Miniconda2-latest-Linux-x86_64.sh (give execution permission fo file)

$ ./Miniconda2-latest-Linux-x86_64.sh

Welcome to Miniconda2 4.6.14

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue

....

Do you wish the installer to initialize Miniconda2
by running conda init? [yes|no]
[no] >>> no

...
```

4. Run miniconda

```
$ ./miniconda2/bin/conda
```

5. Load conda environment

```
. /home/csys/jpina/miniconda2/etc/profile.d/conda.sh
```

6. Load conda environment in your submission script

```
$ cat test_submit.sh

#!/bin/bash
# Load user environment during job excution
#$ -V

# Call parallell environment "mp", and excute in 4 cores
#$ -pe mp 4

# Queue selection
#$ -q hpc

# Load miniconda
.  /home/csys/jpina/miniconda2/etc/profile.d/conda.sh
```

> **NOTE** Loading the conda environment can lead to conflits with the 'module load' command, therefore users should test the environment on a running job when using both conda and modules environments. If possible, use only conda environment.

Conda

Another example of **conda** environment setup.

Login on the submit node

Login on the cluster submission node, check the page *How to Access* for more information:

```
$ ssh -l <username> cirrus.ncg.ingrid.pt
[username@cirrus ~]$ _
```

Prepare a conda virtual environment

The default **python** version for *CentOS 7.x* is ***2.7.5*** which is not suitable for many applications. So, we will create a **python** virtual environment:

```
[username@cirrus ~]$ conda create -n myconda python=3.6
[username@cirrus ~]$ conda activate myconda
```

On the first command, where we create the **conda** virtual environment, you can include a list of applications to include on your environmnet, for example:

```
[username@cirrus ~]$ conda create -n myconda python=3.6 ipython-notebook numpy=1.6
```

Manage the conda virtual environment

It is possible to include additional packages to your **conda** environment, for example:

```
[username@cirrus ~]$ conda activate myconda
[username@cirrus ~]$ conda install numpy
```

You can update your software bandle on the **conda** virtual environment with the command:

```
[username@cirrus ~]$ conda update [scipy ...]
```

or remove a specific application:

```
[username@cirrus ~]$ conda uninstall tensorflow-gpu
```

Check the **conda** help for more information:

```
[username@cirrus ~]$ conda help
[username@cirrus ~]$ conda install --help
```

Manage the conda packages list with pip

It is possible to complemment the **conda** virtual environment packages list with **pip**. For example:

```
[username@cirrus ~]$ conda activate myconda
[username@cirrus ~]$ pip install --upgrade pip
[username@cirrus ~]$ pip install --upgrade setuptools
[username@cirrus ~]$ pip install tensorflow-gpu
[username@cirrus ~]$ pip install keras
```

Manage packages versions

If the applications available on **conda** virtual environment do not match your version requirements you may need to use packages from **pip** repostory; check the availability of ***conda search*** and ***pip search*** command line interfaces.

As an example we have the **tensorflow-gpu** package, when used with **keras**, the **conda** repository downgrades ***tensorflow-gpu** to version *1.15*, but you most likely will prefer version *2.0* . The **pip** repository has the right combination of **tensorflow-gpu** and **keras** packages.

We advise the user to install a package from only one repository in order to guarantee perfect behaviour.

Load conda environment on a batch job

Create a submit script:

```
[username@cirrus ~]$ cat submit.sh

#!/bin/bash

#SBATCH --job-name=MyFirstSlurmJob
#SBATCH --time=0:10:0
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=16

# Be sure to request the correct partition to avoid the job to be held in the queue, furthermore
#□on CIRRUS-B (Minho)  choose for example HPC_4_Days
```

```
#on CIRRUS-A (Lisbon) choose for example hpc
#SBATCH --partition=HPC_4_Days

# check python version
python --version

# Load conda environment
conda activate myconda

# recheck python version
python --version

# your job payload
#....
```

Submit:

```
[username@cirrus ~]$ sbatch submit.sh
Your job 2037792 ("submit.sh") has been submitted
```

After completion:

```
[username@hpc7 ~]$ ls -l
-rwxr-----+ 1 username hpc    668 Jan  7 12:19 submit.sh
-rw-r-----+ 1 username hpc     44 Jan  7 12:18 submit.sh.e2037792
-rw-r-----+ 1 username hpc      0 Jan  7 12:18 submit.sh.o2037792

[username@cirrus ~]$ cat submit.sh.e2037792
Python 2.7.5
Python 3.6.9 :: Anaconda, Inc.
```

Some References

- GitHub OS-agnostic
- Anaconda Documentation
- Conda Documentation

Example of a user application installation

This example will show how to install **Octave** on the hipotetical user *username* home directory **HOME/soft/octave/5.1.0**. The example will use the interactive host to install the software but if you can also write a script and submit a job as long the command line instructions could be automatized.

Login on the submit node

Login on the cluster submition node, check the page *How to Access* for more information:

```
$ ssh -l <username> hpc7.ncg.ingrid.pt
[username@hpc7 ~]$ _
```

Download the source code

```
[username@hpc7 ~]$ wget ftp://ftp.gnu.org/gnu/octave/octave-5.1.0.tar.xz
[username@hpc7 ~]$ tar Jxf octave-5.1.0.tar.xz
[username@hpc7 ~]$ cd octave-5.1.0
```

Configure and install

```
[username@hpc7 ~]$ ./configure --prefix=/home/mygroup/username/soft/octave/5.1.0 --enable-shared
[username@hpc7 ~]$ make
[username@hpc7 ~]$ make check
[username@hpc7 ~]$ make install
```

Setup environment

The most basic would be to configure the appropriate environment variables, or better, create a shell script to load when needed:

```
[username@hpc7 ~]$ cat .octave.bash
export OCTAVE_HOME=/home/mygroup/username/soft/octave/5.1.0
```

```
[ -z "$PATH" ] && export PATH=$OCTAVE_HOME/bin || export PATH=$OCTAVE_HOME/bin:$PATH
[ -z "$CPATH" ] && export PATH=$OCTAVE_HOME/include || export CPATH=$OCTAVE_HOME/include:$CPATH
[ -z "$LD_LIBRARY_PATH" ] && export LD_LIBRARY_PATH=$OCTAVE_HOME/lib || export
LD_LIBRARY_PATH=$OCTAVE_HOME/lib:$LD_LIBRARY_PATH
[ -z "$PKG_CONFIG_PATH" ] && export PAG_CONFIG_PATH=$OCTAVE_HOME/lib/pkgconfig || export
PKG_CONFIG__PATH=$OCTAVE_HOME/lib/pkgconfig:$PKG_CONFIG_PATH
```

Activate environment for application

```
[username@hpc7 ~]$ . .octave.bash
```

Run the application

```
[username@hpc7 ~]$ which octave
~/soft/octave/5.1.0/bin/octave

[username@hpc7 ~]$ octave
octave:1> _
```

Better way to setup environment: USE MODULES

A better way to provide configuration would be using the *module environment tool" customized for the user, check the User Customization With module page. It would be easier to manage and share with other users if needed.

User customization with module

Example of environment configuration for *Octave* application installed on the example ***Example of a User application Installation*** page.

Login on the submit node

Login on the cluster submition node, check the page *How to Access* for more information:

```
$ ssh -l <username> hpc7.ncg.ingrid.pt
[username@hpc7 ~]$ _
```

Select a directory to store your modules environments

In this example we will use **~/.module** on the user *Home* directory:

```
[username@hpc7 ~]$ mkdir .module
```

Create a modules resource file

Create a file named ***~/.modulerc*** with the following content:

```
[username@hpc7 ~]$ cat .modulerc
#%Module1.0#####################################################################
##########
##
## User prefer modules at session startup
##
module use $env(HOME)/.module
```

Create a configuration file for Octave application

Lets create a simple configution file for the **Octave** application installed on the *Home* directory:

```
[username@hpc7 ~]$ mkdir .module/octave
[username@hpc7 ~]$ cat .module/octave/5.1.0
#%Module1.0
##
## octave/5.1.0 modulefile
##

proc ModulesHelp { } {
        global version
        puts stderr "\tSets up Octave"
        puts stderr "\n\tVersion $version\n"
}

module-whatis "Sets up Octave"

set version 5.1.0
set modroot /home/hpc/jmartins/soft/octave/$version

setenv OCTAVE_HOME $modroot
prepend-path PATH $modroot/bin
prepend-path CPATH $modroot/include
prepend-path LD_LIBRARY_PATH $modroot/lib
prepend-path PKG_CONFIG_PATH $modroot/lib/pkgconfig
```

Check the new modules environment

The next time you login on server you will find your environment profile available for normal usage:

```
[username@hpc7 ~]$ module avail
------------------- /home/hpc/jmartins/.module -------------------
octave/5.1.0

------------------- /cvmfs/sw.el7/modules/soft -------------------
aster-13.1.0                gromacs-4.6.7
blat-36.2                   hdf5-1.8.16
....
```

Manage your new modules environment

```
[username@hpc7 ~]$ module load octave/5.1.0

[username@hpc7 ~]$ which octave
/home/mygroup/username/soft/octave/5.1.0

[username@hpc7 ~]$ octave
octave:1> grid□□# a GRID plot will popup
octave:2> exit

[username@hpc7 ~]$ module unload octave/5.1.0

[username@hpc7 ~]$ which octave
<empty>
```

Some Links

- Environment Modules Documentation

udocker Usage Example

Install by your own:

See **IBERGRID 2019 Tuturial**, check also **the general udocker tuturials page**

Use the already available installation:

```
[user@pauli02 ~]$ module avail

------- /cvmfs/sw.el7/modules/soft ----------
gcc-5.5          matlab/R2018a    trimmomatic-0.33
gcc63/mpich-3.2  matlab/R2018b    udocker/1.1.3

[user@pauli02 ~]$ module load udocker

[user@pauli02 ~]$ module list
Currently Loaded Modulefiles:
 1) udocker/1.1.3

```

The ***udocker*** works the same way as the command ***docker***, the big difference is that it runs on *user space* and doesn't require special privileges to run, ***works everywhere***!. The docker hub repository is available and the user is able pull images from the docker hub repository and also his own images.

For example, search for a **Centos** image:

```
[martinsj@pauli02 ~]$ udocker search centos
NAME                            OFFICIAL DESCRIPTION
centos                            [OK] The official build of CentOS.
```

```
pivotaldata/centos-gpdb-dev                 ---- CentOS image for GPDB development. Tag...
pivotaldata/centos-mingw                    ---- Using the mingw toolchain to cross...
```

Pull an image and list it:

```
[martinsj@pauli02 ~]$ udocker pull centos
Downloading layer: sha256:729ec3a6ada3a6d26faca9b4779a037231f1762f759ef34c08bdd61bf52cd704
Downloading layer: sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
Downloading layer: sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4

[martinsj@pauli02 ~]$ udocker images
REPOSITORY
centos:latest
```

Create and start a container:

```
[martinsj@pauli02 ~]$ udocker create --name=MY_CENTOS  centos
abfe7e30-d25d-3d47-bfc3-ff11401bb430

[martinsj@pauli02 ~]$ udocker run --volume=$HOME --workdir=$HOME --user=$USER MY_CENTOS bash -li

 ********************************************************************************
 *                                                              *
 *             STARTING abfe7e30-d25d-3d47-bfc3-ff11401bb430              *
 *                                                              *
 ********************************************************************************
 executing: bash

abfe7e30$ ls -l
total 183852
drwxr-xr-x  2 martinsj csys         6 Mar 21  2018 Desktop
drwxr-xr-x  8 martinsj csys        88 Apr 17  2018 Template
-rwxr--r--  1 martinsj csys         6 Nov  1  2018 VERSION
...
```

Inline general help is available

```
[martinsj@pauli02 ~]$ udocker help
    Syntax:
      udocker  <command>  [command_options]  <command_args>

    Commands:
      search <repo/image:tag>       :Search dockerhub for container images
      pull <repo/image:tag>         :Pull container image from dockerhub
      images                        :List container images
      create <repo/image:tag>       :Create container from a pulled image
      ps                            :List created containers
      ...
```

as long as a subcommand specialized help

```
[martinsj@pauli02 ~]$ udocker run --help

    run: execute a container
    run [options] <container-id-or-name>
    run [options] <repo/image:tag>
    --rm                        :delete container upon exit
    --workdir=/home/userXX      :working directory set to /home/userXX
    --user=userXX               :run as userXX
    --user=root                 :run as root
    --volume=/data:/mnt         :mount host directory /data in /mnt
    ...
```

How to submit a job that uses TensorFlow

With this tutorial you will be able to submit a job that uses TensorFlow to the batch cluster.

The following steps allow the user to execute a Python script that uses TensorFlow and other Python libraries.

Copy the project folder to the cluster

```
[user@fedora ~]$ scp -r -J user@fw03 /home/user/my_project/ user@cirrus01
```

Access the cluster

```
[user@fedora ~]$ ssh user@cirrus01
```

Clone the reference repository

```
[user@cirrus01]$ git clone https://gitlab.com/lip-computing/computing/tf_run_job.git
```

Submit the job with the Python script inside project folder. In this example, the datasets are in my_datasets subfolder.

```
[user@cirrus01]$ cd my_project
[user@cirrus01 my_project]$ sbatch ~/tf_run_job/run_job --input my_python_script.py --file
my_datasets/dataset1.csv my_datasets/dataset2.csv
```

Once the job is completed the console log with the program messages will be written to a folder in the user's home directory.

```
[user@cirrus01 my_project]$ cat slurm-124811.out
* ------------------------------------------------------------
* Running PROLOG for run_job on Tue Nov 17 17:22:01 WET 2020
*    PARTITION             : gpu
*    JOB_NAME              : run_job
*    JOB_ID                : 124811
```

```
*   USER                : user
*   NODE_LIST           : hpc050
*   SLURM_NNODES        : 1
*   SLURM_NPROCS        :
*   SLURM_NTASKS        :
*   SLURM_JOB_CPUS_PER_NODE : 1
*   WORK_DIR            : /users/hpc/user/my_project
* -----------------------------------------------------------
Info: deleting container: 61fb9513-b33d-3b7f-85ed-25db26202b61
7f5d9200-712f-3134-a470-defdffb21e81
Warning: non-existing user will be created



################################################################################
############
 #                                                #
 #           STARTING 7f5d9200-712f-3134-a470-defdffb21e81          #
 #                                                #

################################################################################
############
 executing: bash
Results available on workdir: /home/hpc/user/Job.ZlV3RW
```

Any additional support for this procedure or to use different requirements for the provided TensorFlow docker image with GPU, just contact helpdesk@incd.pt.

Intel MKL

Intel Math Kernel Library (Intel MKL) is a library of optimized math routines for science, engineering, and financial applications. Core math functions include BLAS, LAPACK, ScaLAPACK, sparse solvers, fast Fourier transforms, and vector math. The routines in MKL are hand-optimized specifically for Intel processors.

Documentation

The reference manual for INTEL MKL may be found here.

It includes:

- **BLAS (Basic Linear Algebra Subprograms) and Sparse BLAS Routines** - Sparse Basic Linear Algebra Subprograms (BLAS) perform vector and matrix operations similar to BLAS Level 1, 2, and 3 routines. Sparse BLAS routines take advantage of vector and matrix sparsity: they allow you to store only non-zero elements of vectors and matrices.
  - BLAS Level 1 Routines and Functions (vector-vector operations)
  - BLAS Level 2 Routines (matrix-vector operations)
  - BLAS Level 3 Routines (matrix-matrix operations)
  - Sparse BLAS Level 1 Routines and Functions (vector-vector operations).
  - Sparse BLAS Level 2 and Level 3 (matrix-vector and matrix-matrix operations)
- **LAPACK Routines** - used for solving systems of linear equations and performing a number of related computational tasks.The library includes LAPACK routines for both real and complex data. Routines are supported for systems of equations with the following types of matrices:
  - general
  - banded
  - symmetric or Hermitian positive-definite (both full and packed storage)
  - symmetric or Hermitian positive-definite banded
  - symmetric or Hermitian indefinite (both full and packed storage)
  - symmetric or Hermitian indefinite banded
  - triangular (both full and packed storage)
  - triangular banded
  - tridiagonal.
    - For each of the above matrix types, the library includes routines for performing the following computations:
      - factoring the matrix (except for triangular matrices)
      - equilibrating the matrix
      - solving a system of linear equations

    - estimating the condition number of a matrix
    - refining the solution of linear equations and computing its error bounds
    - inverting the matrix.
- **ScaLAPACK Routines** - Routines are supported for both real and complex dense and band matrices to perform the tasks of solving systems of linear equations, solving linear least-squares problems, eigenvalue and singular value problems, as well as performing a number of related computational tasks. All routines are available in both single precision and double precision.
- **Vector Mathematical Functions**
    - sin
    - tan
    - ...
- **Statistical Functions**
    - RNG
    - Convolution and Correlation
- **Fourier Transform Functions**
    - DFT Functions
    - Cluster DFT Funtions - this library was designed to perform Discrete Fourier Transform on a cluster, that is, a group of computers interconnected via a network. Each computer (node) in the cluster has its own memory and processor(s). Data interchanges between the nodes are provided by the network. To organize communication between different processes, the cluster DFT function library uses Message Passing Interface (MPI). Given the number of available MPI implementations (for example, MPICH, Intel® MPI and others), Cluster DFT works with MPI via a message-passing library for linear algebra, called BLACS, to avoid dependence on a specific MPI implementation.

Benchmarks

These benchmarks are offered to help you make informed decisions about which routines to use in your applications, including performance for each major function domain in Intel® oneAPI Math Kernel Library (oneMKL) by processor family. Some benchmark charts only include absolute performance measurements for specific problem sizes. Others compare previous versions, popular alternate open-source libraries, and other functions for oneMKL [2].

image-1611056010894.png
Image not found or type unknown

image-1611056010898.png
Image not found or type unknown

Why is Intel MKL faster?

Optimization done for maximum speed. Resource limited optimization – exhaust one or more resource of system [3]:

- **CPU**: Register use, FP units.
- **Cache**: Keep data in cache as long as possible; deal with cache interleaving.
- **TLBs**: Maximally use data on each page.
- **Memory bandwidth**: Minimally access memory.
- **Computer**: Use all the processor cores available using threading.
- **System**: Use all the nodes available.

Compilation

Compile with `intel/2020`

```
#Environment setup
module purge
module load intel/2020
module load intel/2020.mkl
source /cvmfs/sw.el7/intel/2020/mkl/bin/mklvars.sh intel64

icc -mkl <source_file.c> -o <output_binary_name>

./<output_binary_name> #Execute binary
```

Compile with `intel/mvapich2/2.3.3`

```
#Environment setup
module purge
module load intel/2020
module load intel/2020.mkl
module load intel/mvapich2/2.3.3
source /cvmfs/sw.el7/intel/2020/mkl/bin/mklvars.sh intel64

mpicc -mkl <source_file.c> -o <output_binary_name>

./<output_binary_name> #Execute binary
```

Compile with `gcc-8.1`

```
#Environment setup
module purge
module load gcc-8.1
```

```
module load intel/2020.mkl
source /cvmfs/sw.el7/intel/2020/mkl/bin/mklvars.sh intel64

#Program compile
gcc -L${MKLROOT}/lib/intel64 -Wl,--no-as-needed -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread -lm -ldl  <source_file.c>  -o <output_binary_name>

#Execute binary
./<output_binary_name>
```

Performance Test

To test performance, we start by running an example and perform the following calculation: `C = alpha*A*B + C` where A, B and C are matrices of the same dimension.

WITH MKL

	GCC	MPICC	ICC
n = 2000	0.19 s	0.14 s	0.16 s
n = 20000	51.86 s	50.01 s	49.71 s

WITH MKL AND MPI

	1 Node	2 Nodes	3 Nodes
MVAPICH2			
MPICH			
INTEL MPI			

References

[1] https://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/mkl_lapack_examples/c_bindings.htm

[2] https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html

[3] intel.cn/content/dam/www/public/apac/xa/en/pdfs/ssg/Intel_Performance_Libraries_Intel_Math_Kernel_Library(MKL).pdf

Workflow using git

Install git on your machine:

See **Git installation guide.**

Create and setup your gitlab account:

To create a gitlab acount see **Gitlab Sign Up.**

In the settings tab add your SSH key to your gitlab account. If you don't have a ssh key see **Learn how to create a SSH key.**

Follow the workflow

Clone the remote repository into a new local directory.

```
[user@fedora ~]$ mkdir my_repo
[user@fedora ~]$ cd my_repo
[user@fedora my_repo]$ git clone git@git01.ncg.ingrid.pt:lip-computing/project_name.git
```

Create a new branch to work on a new feature. The branch is named new_feature in the example bellow.

```
[user@fedora my_repo]$ cd project_name
[user@fedora project_name]$ git branch new_feature
[user@fedora project_name]$ git checkout new_feature
Switched to branch 'new_feature'
```

Push your changes to the remote repository. In the following example, new_feature.py is the file that contains the code for the new feature.

```
[user@fedora project_name]$ git add new_feature.py
[user@fedora project_name]$ git status
On branch new_feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   new_feature.py
```

At this point your changes were added to the staging area. First commit your changes and push them to the remote repository so that other team members can review them.

```
[user@fedora project_name]$ git commit -m "<your_commit_message>"
[new_feature f8ebb26] <your_commit_message>
 Author: User <user@lip.pt>
 1 file changed, 1 insertion(+), 1 deletion(-)
```

```
[user@fedora project_name]$ git push origin new_feature
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 625 bytes | 625.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for new_feature, visit:
remote:   https://git01.ncg.ingrid.pt:user/lip-computing/project_name/-
/merge_requests/new?merge_request%5Bsource_branch%5D=new_feature
remote:
To git01.ncg.ingrid.pt:user/lip-computing/project_name.git
 * [new branch]      new_feature -> new_feature
```

Your branch is now available in the remote repository. From the dashboard you can create a merge request and assign a team member to review your code.

Once your code has been reviewed, all the changes to your code have been performed and the final version has been approved, your branch can be merged to the master branch.

You can now update your local repository to the latest state of the remote repository and work on another feature repeating the same steps.

```
[user@fedora project_name]$ git fetch
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 250 bytes | 250.00 KiB/s, done.
From git01.ncg.ingrid.pt:user/lip-computing/project_name
   113c798..e490c8f  master     -> origin/master
[user@fedora project_name]$ git merge -X theirs origin/master
```

```
Updating f8ebb26..e490c8f
Fast-forward
```

Manage conflicts

A conflict arises if the state of the remote repository changed while you were working on your local repository. This means you don't have the latest state of the remote repository on your local machine.

```
[user@fedora test]$ git push origin new_feature
To git01.ncg.ingrid.pt:user/lip-computing/project_name.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://git@git01.ncg.ingrid.pt:lip-computing/project_name.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

AlphaFold

1. Introduction

The **INCD** team prepared a local installation of AlphaPhold using a container based on **UDOCKER** (instead of **DOCKER**) and includes the *Genetic Database*.

The local installation provide the **AlphaFold** version **2.1.1** over a container based on **Ubuntu 18.04** distribution with **cuda-11.0** and **cudnn-8**.

The main resource target of **AlphaFold** is the **GPU** but the application also execute only on the **CPU** although the performance is substantially worst, see the Benchmarks section bellow.

1.1 Environment

The environment is activate with command

```
$ module load udocker/alphaphold/2.1.1
```

this will activate automatically a virtual environment ready to start the **AlphaFold** container throught the python script **run_udocker.py**.

1.2 Data Base Location

The **Genetic Database** is installed bellow the filesystem directory

```
/users3/data/alphafold
```

on read-only mode, upgrades may be requested using the *helpdesk@incd.pt* address.

1.3 run_udocker.py Script

The **run_udocker.py** script was adapted from the **run_docker.py** script normally used by **AlphaFold** with the **docker** container technology.

The **run_udocker.py** accept the same options as the **run_docker.py** script with a few minor changes that we hope it will facilitate user interaction. The user may change the script behavour throught environment variables or command line options, we can see only the changes bellow:

Optional environment variables:

Variable Name	Default Value	Comment
DOWNLOAD_DIR	none	Genetic database location (absolute path)
OUPTPUT_DIR	none	Output results directory (absolute path)

Command line options:

Command Option	Mandatory	Default Value	Comment
--data_dir	no	**/local/alphafold** or **/users3/data/alphafold**	Genetic database location, takes precedence over DOWNLOAD_DIR when both are selected
--output_dir	no	<working_dir>/output	Absolute path to the results directory, takes precedence over OUTPUT_DIR when both are selected

> The option **--data_dir** is required on the standard AlphaFold **run_docker.py** script, we choose to select automatically the location of the **genetic database** but the user may change this path throught the environment variable **DOWNLOAD_DIR** or the command line option **--data_dir**. When possible, we provide a local copy to the workernodes of the database directory in order to improve job performance.

> The AlphaFold standard output results directory location is **/tmp/alphafold** by default, please note that we change this location to the local working directory, the user can select a different path throught the environment variable **OUTPUT_DIR** or the command line option **--output_dir**.

2. How to Run

We only need a protein and a submition script, if we analyze multiple proteins on parallel it is advise to submit then from different directory in order to avoid interference between runs.

2.1 Example on Partition "gpu"

Lets analyze the https://www.uniprot.org/uniprot/P19113 protein, for example.

Create a working directory and get the protein:

```
[user@cirrus ~]$ mkdir run_P19113
[user@cirrus ~]$ cd run_P19113
[user@cirrus run_P19113]$ wget -q https://www.uniprot.org/uniprot/P19113.fasta
```

Use your favority editor the create the submition script **submit.sh***:

```
[user@cirrus run_P19113]$ emacs submit.sh
#!/bin/bash
# -----------------------------------------------------------------------------
#SBATCH --job-name=P19113
#SBATCH --partition=gpu
#SBATCH --mem=50G
#SBATCH --ntasks=4
#SBATCH --gres=gpu
# -----------------------------------------------------------------------------
module purge
module load udocker/alphafold/2.1.1
run_udocker.py --fasta_paths=P19113.fasta --max_template_date=2020-05-14
```

Finally, submit your job, check if it is running and wait for it:

```
[user@cirrus run_P19113]$ sbatch submit.sh
[user@cirrus run_P19113]$ squeue
```

When finish the local directory **./output** will have the analyze results.

2.2 Example on Partition "fct"

```
[user@cirrus run_P19113]$ emacs submit.sh
#!/bin/bash
# ------------------------------------------------------------------------------
#SBATCH --job-name=P19113
#SBATCH --partition=fct
#SBATCH --qos=<qos>
#SBATCH --account=<account>□□# optional on most cases
#SBATCH --mem=50G
#SBATCH --ntasks=4
#SBATCH --gres=gpu
# ------------------------------------------------------------------------------
module purge
module load udocker/alphafold/2.1.1
run_udocker.py --fasta_paths=P19113.fasta --max_template_date=2020-05-14
```

2.3 Example on Partition "hpc"

```
[user@cirrus run_P19113]$ emacs submit.sh
#!/bin/bash
# ------------------------------------------------------------------------------
#SBATCH --job-name=P19113
#SBATCH --partition=hpc
#SBATCH --mem=50G
#SBATCH --ntasks=4
# ------------------------------------------------------------------------------
module purge
module load udocker/alphafold/2.1.1
run_udocker.py --fasta_paths=P19113.fasta --max_template_date=2020-05-14
```

2.4 sbatch Options

--partition=XX

The best job performance is achivied on the **gpu** or **fct** partitions, the later is restricted to users with a valid **QOS**.

The **alphafold** and also run on the **hpc** partition but is this case it will use only a slower **CPU** and there is no **GPU** available, the total run time is roughly eight times greather when compared to jobs executed on the **gpu** or **fct** partitions.

--mem=50G

The default job memory allocation per cpu depends on the used partition but it may be insuficient, we recommend you to request **50GB** of memory, the benchmarks sugest this value should be enough on all cases.

--ntasks=4

Apparentelly this is the maximum number of tasks needed by the application, we didn't get any noticible improvement when rising this parameter.

--gres=gpu

The partitions **gpu** and **fct** provide up to eight **GPUs**. The application was built for compute using **GPU**, there is no point is requesting more than one **GPU**, we didn't notice any improvement on the total run time. We also notice that the total compute time for both types of available **GPUs** is similar.

The **alphafold** also run only on **CPU** but the total run time increase substantial, as seen on benchmarks results bellow.

4. Benchmarks

We made some benchmarks with the protein *P19113* in order to help users organizing their work.

The results bellow sugest that the best choice would be use four **CPU** tasks, one **GPU** and let the system select the local copy of the *genetic data base* on the workernodes.

Since a **GPU** run takes roughly two hours and half then users may run up to thirty five protein analyzes in one submit job, as long they are executed in sequence.

Partition	CPU	#CPU	GPU	#GPU	#JOBS	DOWNLOAD_DIR	ELAPSED_TIME
gpu/fct	EPYC_7552	4	Tesla_T4	1	1	/local/alphafold	02:22:19
gpu/fct	EPYC_7552	4	Tesla_V100S	1	1	/local/alphafold	02:38:21
gpu/fct	EPYC_7552	4	Tesla_T4	2	1	/local/alphafold	02:22:25

gpu/fct	EPYC_7552	4	Tesla_T4	1	1	/users3/data/alphafold	15:59:50
gpu/fct	EPYC_7552	4	Tesla_V100S	1	1	/users3/data/alphafold	11:40:04
gpu/fct	EPYC_7552	4	Tesla_T4	2	1	/users3/data/alphafold	14:58:52
gpu/fct	EPYC_7552	4		0	1	/local/alphafold	16:17:32
gpu/fct	EPYC_7552	4		0	1	/users3/data/alphafold	18:22:07
gpu/fct	EPYC_7552	4		0	4	/local/alphafold	17:53:25
hpc	EPYC_7501	4		0	3	/local/alphafold	21:44:35
hpc	EPYC_7501	32		0	1	/local/alphafold	16:35:59
hpc	EPYC_7501	4		0	1	/users3/data/alphafold	1-02:28:33
hpc	EPYC_7501	16		0	1	/users3/data/alphafold	1-03:42:23
hpc	EPYC_7501	32		0	1	/users3/data/alphafold	1-03:15:19

5. References

- https://github.com/deepmind/alphafold
- https://github.com/indigo-dc/udocker
- https://www.docker.com
- https://www.uniprot.org/uniprot
- https://www.uniprot.org/uniprot/P19113

AlphaFold3

1. Introduction

This package provides an implementation of the inference pipeline of AlphaFold 3. See below for how to access the model parameters. You may only use AlphaFold 3 model parameters if received directly from Google. Use is subject to these terms of use.

Any publication that discloses findings arising from using this source code, the model parameters or outputs produced by those should cite the Accurate structure prediction of biomolecular interactions with AlphaFold 3 paper.

Please also refer to the Supplementary Information for a detailed description of the method.

AlphaFold 3 is also available at alphafoldserver.com for non-commercial use, though with a more limited set of ligands and covalent modifications.

If you have any questions, please contact the AlphaFold team at alphafold@google.com.

1.1 Local Installation

The **CNCA** team prepared a local installation of AlphaPhold3 using a container based on **singularity or (appatainer)** including the *Genetic Database*.

> The *Model Parameters* is not included, users must request access filling a form and comply with the Google DeepMind terms of use at all times.

The local installation provide the **AlphaFold3** version **3.0.1** over a container based on **Ubuntu 24.04** distribution with **cuda-12.6**. The container is already prepared and is used throughtout a wrapper that accept all *run_alphafold.py* options; the *Genetic Database* location is already configured, users should provide the json path, model and output directories location.

The main resource target of **AlphaFold3** is the **GPU** but the application can execute the data stage only on the **CPU** although the performance is substantially worst. The model inference stage requires a GPU, if you submit a job to a partition without a **GPU**, such as the **hpc** or **fct** partitions, then the option ***--norun-inference*** is added automatically to the *run_alphafold.py* script.

The user is free to prepare is own container as explained on https://github.com/google-deepmind/alphafold3/tree/main.

1.2 Environment

The environment is activate with command

```
$ module load alphaphold3/3.0.1
```

this will activate automatically a virtual environment ready to start the **AlphaFold3** container throught the python script *run_alphafold.py*.

The command *run_alphafold.py* is a wrapper to the real script in the container and accept the same options. For example, you can get the *run_alphafold.py* help with command

```
$ run_alphafold.py --helpshort
```

1.3 Data Base Location

The **Genetic Database** is installed only on local disk of workernodes bellow the directory

```
/local/alphafold3
```

The environment variable **ALPHAFOLD3_DATABASE** points to this location.

> The **Genectic Database** is not available on the user interfaces.

2. How to Run

2.1 Choose a working directory

```
$ mkdir ~/alphafold3_test
$ cd ~/alphafold3_test
```

2.2 Obtain the Model Parameters

The model parameters access is subject to Google DeepMind terms of use, please fill the form https://forms.gle/svvpY4u2jsHEwWYS6 and follow the instructions. Once you have access to the model parameters you will be responsible to keep the data private.

2.3 Create a submission script

Choose your favority editor and create a file, for example *alphafold3.sh*, with the following content:

```
$ emacs alphafold3.sh
#!/bin/bash
#SBATCH --partition=gpu
#SBATCH --gres=gpu
#SBATCH --nodes=1
#SBATCH --ntasks=8
#SBATCH --mem=64G

module purge
module load alphafold3
run_alphafold.py \
    --json_path=fold_input.json \
    --model_dir=./models \
    --output_dir=./af_output
```

Create the *models* local directory and copy the model parameters into it:

```
$ mkdir models
```

You can place the model parameters on any other directory you just have to insert the appropriate directory location on the *--model_dir* option.

It is recommended to request at least 64GB of RAM but you can increase if necessary.

The output directory is created automatically if it does not exist.

2.4 Create the input JSON

You can use the example of section **"Installation and Running Your First Prediction"** of page https://github.com/google-deepmind/alphafold3?tab=readme-ov-file.

```
$ emacs fold_input.json
{
"name": "2PV7",
"sequences": [
  {
    "protein": {
      "id": ["A", "B"],
          "sequence"
"GMRESYANENQFGFKTINSDIHKIVIVGGYGKLGGLFARYLRASGYPISILDREDWAVAESILANADVVIVSVPINLTLETIERLKPYL
TENMLLADLTSVKREPLAKMLEVHTGAVLGLHPMFGADIASMAKQVVVRCDGRFPERYEWLLEQIQIWGAKIYQTNATEHDHNM
TYIQALRHFSTFANGLHLSKQPINLANLLALSSPIYRLELAMIGRLFAQDAELYADIIMDKSENLAVIETLKQTYDEALTFFENNDRQG
FIDAFHKVRDWFGDYSEQFLKESRQLLQQANDLKQG"
    }
  }
],
"modelSeeds": [1],
"dialect": "alphafold3",
"version": 1
}
```

2.5 Submit the job and check status

```
$ sbatch alphafold3.sh
Submitted batch job 22719987

$ squeue
JOBID    PARTITION NAME      USER     ST TIME NODES CPUS TRES_PER_NODE NODELIST
22719987 gpu       alphafold3 martinsj PD 0:00 1     8    gres/gpu
```

The *squeue* command list shows that the job is pending, or waiting for resources.

2.6 Check the running job

Eventually the job will run and the *squeue* command will show something like:

```
$ squeue
JOBID    PARTITION NAME       USER     ST TIME NODES CPUS TRES_PER_NODE NODELIST
22719987 gpu       alphafold3 martinsj R  1:25 1     8    gres/gpu      hpc060
```

You can get more details about the job with command *pestat*:

```
$ pestat -j 22719987
Select only nodes with jobs in joblist=22719987
Hostname Partition  Node Num_CPU CPUload Memsize Freemem  GRES/       Joblist
                State Use/Tot           (MB)    (MB)  node        JobId User GRES/job ...
hpc060   gpu        mix   8  96   2.05*  512000   28466* gpu:t4:3(S:0-1) 22719987 martinsj
```

2.7 Check the finished job

On completion the job vanish from *squeue* list and you have the output on subdirectory "af_output" and the job standard output will be on file *slurm-2271987.out*:

```
$ ls -l
drwxr-x---+ 2 martinsj csys  4096 out  4 14:08 af_input
drwxr-x---+ 2 martinsj csys  4096 out  4 14:08 models
drwxrwx---+ 4 martinsj csys  4096 out  4 16:10 af_output
-rw-rw----+ 1 martinsj csys   424 out  4 14:29 alphafold3.sh
-rw-rw----+ 1 martinsj csys 46591 out  4 16:11 slurm-22719987.out

$ ls -l af_output/2PV7/
-rw-rw----+ 1 martinsj csys 3118163 Oct  4 16:11 2PV7_confidences.json
-rw-rw----+ 1 martinsj csys 7301275 Oct  4 16:11 2PV7_data.json
-rw-rw----+ 1 martinsj csys  409731 Oct  4 16:11 2PV7_model.cif
-rw-rw----+ 1 martinsj csys     147 Oct  4 16:11 2PV7_ranking_scores.csv
-rw-rw----+ 1 martinsj csys     332 Oct  4 16:11 2PV7_summary_confidences.json
drwxrwx---+ 2 martinsj csys    4096 Oct  4 16:11 seed-1_sample-0
drwxrwx---+ 2 martinsj csys    4096 Oct  4 16:11 seed-1_sample-1
drwxrwx---+ 2 martinsj csys    4096 Oct  4 16:11 seed-1_sample-2
drwxrwx---+ 2 martinsj csys    4096 Oct  4 16:11 seed-1_sample-3
drwxrwx---+ 2 martinsj csys    4096 Oct  4 16:11 seed-1_sample-4
```

```
-rw-rw----+ 1 martinsj csys   13036 Oct  4 16:11 TERMS_OF_USE.md
```

2.8 Get some job statistics

You can get some job statistics with commands:

```
$ seff 22719987
Job ID: 22719987
Cluster: production
User/Group: martinsj/csys
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 8
CPU Utilized: 01:03:58
CPU Efficiency: 44.01% of 02:25:20 core-walltime
Job Wall-clock time: 00:18:10
Memory Utilized: 6.20 GB
Memory Efficiency: 6.46% of 96.00 GB
```

and

```
$ sacct -j 22719987
JobID           JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
22719987     alphafold+       gpu       csys         8  COMPLETED      0:0
22719987.ba+      batch                 csys         8  COMPLETED      0:0
22719987.ex+     extern                 csys         8  COMPLETED      0:0
```

You can obtain more details with command

```
$ macct -j 22719987
```

3. Permance

It is recommended to request a *NVIDIA A100 80GB* GPU or a GPU with Compute Capatibily of *8.0* at least. The container is built with the *NVIDIA A100 80GB* in mind.

The **CIRRUS** provide three GPU types:

- NVIDIA A100 80GB
- NVIDIA Tesla V100s 32GB

- NVIDIA Tesla T4 16GB

The best performance will be obtained with the *NVIDIA A100 80GB* controller but this is the most requested resource and users may have to wait days to obtain one. The other controllers are less performant but more available and it may compensate to use the *Tesla* GPU's. As a reference, the example above completation times for each GPU was the following:

Controller	Completion Time (hh:mm:ss)
NVIDIA A100	18:10
NVIDIA Tesla V100s	2:19:31
NVIDIA Tesla T4	3:02:37

Users may request a specific controller modifying the option **SBATCH --gres=gpu** on the script to:

- #SBATCH --gres=gpu:a100
- #SBATCH --gres=gpu:v100s
- #SBATCH --gres=gpu:t4

to request the *A100*, or the *Tesla V100s*, or the *Tesla T4*, respectively. The *Tesla* GPU's controllers Compute Capabilities are less than *8.0* but it is possible to run *alphafold3*. The *run_alphafold.py* wrapper insert the right options on the container *run_alphafold.py* script in order to execute, although with reduced performance, on the *Tesla* GPU's.

4. References

- AlphaFold3
- Installation and Running Your First Prediction
- Request to access model parameters for AlphFold3