Compiling on Atos HPCF and ECS

Xavier Abellan

xavier.abellan@ecmwf.int

ECMWF

Do you need to compile?

Toolchains

- Several compiler suites available:
  - GCC: 8, 9, 10, 11 and 12
  - Intel: 2021.4
  - AMD AOCC 3.1 and 4.0
  - NVIDIA HPC SDK (former PGI)
- Several MPI implementations
  - OpenMPI 4
  - Intel MPI 2021
  - HPCX OpenMPI (based on OpenMPI 4)

Recommended toolchain for stability and performance:

intel 2021.4

hpcx-openmpi 2.9.0

intel-mkl 19.0.5

Environment Modules

- Working with different toolchains: the **prgenv** module
  - Active toolchain loaded
  - Affects what modules are “loadable”
  - Ensures that sensitive packages are loaded with the desired “flavour” to avoid conflicts
    - Loading a different prgenv will reload all required modules automatically
  - It allows you to load secondary compilers of different family without affecting the whole stack

```
[usxa@aa6-100 ~]$ module avail prgenv


------------------------------------------------------ Global Aliases ------------------------------------------------------
   pa -> prgenv/amd    pe -> prgenv/expert    pg -> prgenv/gnu    pi -> prgenv/intel    pn -> prgenv/nvidia

---------------------------------------- /usr/local/apps/modulefiles/lmod/prgenvs ----------------------------------------
   conda/4.10.1    prgenv/amd (a)    prgenv/expert (e)    prgenv/gnu (L,D:g)    prgenv/intel (i)    prgenv/nvidia (n)
```

https://confluence.ecmwf.int/x/eA6UCg

Environment Modules

```
[usxa@aa6-100 ~]$ module show netcdf4
-------------------------------------------------------------------------------------------------------------------------------------------
   /usr/local/apps/modulefiles/lmod/flavours/gnu/8.4/netcdf4/4.7.4.lua:
-------------------------------------------------------------------------------------------------------------------------------------------
setenv("netcdf4_DIR","/usr/local/apps/netcdf4/4.7.4/GNU/8.4")
setenv("NETCDF4_DIR","/usr/local/apps/netcdf4/4.7.4/GNU/8.4")
setenv("netcdf4_VERSION","4.7.4")
setenv("NETCDF4_VERSION","4.7.4")
family("netcdf")
prepend_path("PATH","/usr/local/apps/netcdf4/4.7.4/GNU/8.4/bin")
prepend_path("PKG_CONFIG_PATH","/usr/local/apps/netcdf4/4.7.4/GNU/8.4/lib/pkgconfig")
prepend_path("MANPATH","/usr/local/apps/netcdf4/4.7.4/GNU/8.4/share/man")
prepend_path("INFOPATH","/usr/local/apps/netcdf4/4.7.4/GNU/8.4/share/info")
setenv("NETCDF4_LIB","-L/usr/local/apps/netcdf4/4.7.4/GNU/8.4/lib -Wl,-rpath,/usr/local/apps/netcdf4/4.7.4/GNU/8.4/lib -lnetcdff -lnetcdf_c++ -lnetcdf"
)
setenv("NETCDF4_INCLUDE","-I/usr/local/apps/netcdf4/4.7.4/GNU/8.4/include")
whatis("NetCDF (Network Common Data Form) is a set of software libraries and machine-independent data formats that support the creation, access, and sh
aring of array-oriented scientific data.")
help([[NetCDF (Network Common Data Form) is a set of software libraries and machine-independent data formats that support the creation, access, and sha
ring of array-oriented scientific data. It is also a community standard for sharing scientific data. The Unidata Program Center supports and maintains
netCDF programming interfaces for C, C++, Java, and Fortran. Programming interfaces are also available for Python, IDL, MATLAB, R, Ruby, and Perl.
For more information visit https://www.unidata.ucar.edu/software/netcdf/
]])

[usxa@aa6-100 ~]$ module load netcdf4
[usxa@aa6-100 ~]$ ml

Currently Loaded Modules:
  1) gcc/8.4.1   2) prgenv/gnu   3) netcdf4/4.7.4



[usxa@aa6-100 ~]$ which nc-config
/usr/local/apps/netcdf4/4.7.4/GNU/8.4/bin/nc-config
[usxa@aa6-100 ~]$
```

Compiler names

Family	Language	Command
GNU	C	gcc
	C++	g++
	Fortran 77/90	gfortran
INTEL	C	icc
	C++	icpc
	Fortran 77/90	ifort
AMD AOCC	C	clang
	C++	clang++
	Fortran	flang
NVIDIA	C	pgcc
	C++	pgCC
	Fortran 77	pgf77
	Fortran 90	pgf90

- No compiler wrappers (`cc, CC, ftn…`)
  - use compilers directly (`gcc, icc…`)
  - use environment variables:

    `$CC, $CXX, $FC`
- Different compilers may use slightly different options

https://confluence.ecmwf.int/x/Xh6jAg

Software Stack: Building Software

- Flags for module-loaded libraries will not be added automatically!
  - use environment variables provided by modules

```
$ module show netcdf4 | egrep "DIR|INCLUDE|LIB"
setenv("netcdf4_DIR","/usr/local/apps/netcdf4/4.7.4/GNU/8.3")
setenv("NETCDF4_DIR","/usr/local/apps/netcdf4/4.7.4/GNU/8.3")
setenv("NETCDF4_LIB","-L/usr/local/apps/netcdf4/4.7.4/GNU/8.3/lib
       -Wl,-rpath,/usr/local/apps/netcdf4/4.7.4/GNU/8.3/lib -lnetcdff -lnetcdf_c++ -lnetcdf")
setenv("NETCDF4_INCLUDE","-I/usr/local/apps/netcdf4/4.7.4/GNU/8.3/include")
```

https://confluence.ecmwf.int/x/XxhbDg

Step 1: Considerations before you start

- Where to build:
  - In general: use of PERM space is a good idea as a working build directory
  - For small builds: $TMPDIR may be used to speed it up.
  - **Avoid using SCRATCH**: very slow for compilation.
- Where to install your own software:
  - PERM recommended. For example:

```
$PERM/apps/yourpackage/yourversion/
```

  - Home is discouraged: it may not have enough space.
  - **Avoid using SCRATCH**: contents may be deleted automatically.

Step 1: Considerations before you start

- Be aware of the environment.
  - Compiler
  - 3rd party libraries and tools

```
$ module list
```

- Build using the batch system.
  - Interactive nodes may have stricter limits that may have a negative impact
  - Run interactively only for small debugging or troubleshooting.

Step 2: Establishing a plan

- If it's your own program…
  - Consider using ***make***, even if your program is just in one or a few source files

```
EXEC = myprogram
CFLAGS = -O3 $(ECCODES_INCLUDE)
LDFLAGS = $(ECCODES_LIB)
CC = gcc

src = $(wildcard *.c)
obj = $(src:.c=.o)

all: $(EXEC)

%o: %.c
     $(CC) $(CFLAGS) -o $@ $<

$(EXEC): $(obj)
    $(CC) -o $@ $^ $(LDFLAGS)

.PHONY: clean
clean:
    rm -f $(obj) $(EXEC)
```

Step 2: Establishing a plan

- If it’s someone else’s software, program, library…
  - **Read its documentation first!**
    - README, INSTALL, doc/*
    - Official documentation on their website
  - What build method ?
    - GNU Autotools
    - Cmake
    - Plain Makefiles
    - Others / Custom…

Step 3: GNU Autotools

- Quite common in Linux
  - `configure - make - (make check) - make install`
- Run `./configure --help`
  - `--prefix=$PERM/apps/myprogram/myversion`
  - Pay attention to required/optional switches, especially for dependencies. i.e.
    - `--with-netcdf=$NETCDF_DIR`
- Load all the necessary modules.
  - **Never hardcode a path! Use environment variables provided by modules**

Step 3: GNU Autotools

- In some cases, it may be necessary to define certain environment variables, i.e.

```
export CFLAGS="-I$NETCDF_DIR/include
export LDFLAGS="-L$NETCDF_DIR/lib -Wl,-rpath,$NETCDF_DIR/lib"
```

- If ***configure*** fails, check the file config.log
  - Look for the failed test, it may give you a clue of what is missing

Step 3: CMake

- Widely used, growing popularity.
- Build directory detached from source
- Configuration done by defining variables
  - `-DCMAKE_BUILD_TYPE`
  - `-DCMAKE_INSTALL_PREFIX`
  - `-DCMAKE_INSTALL_RPATH`
  - `-DENABLE_XXX=ON/OFF`
- Every package has different options…
  - check its documentation!

```
mkdir build
cd build

cmake  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_INSTALL_PREFIX=/path \
  -DCMAKE_INSTALL_RPATH=/path/lib \
  -DENABLE_NETCDF=ON \
  ..

make
make install
```

Step 3: CMake

- If ***cmake*** fails…
  - Check CMakeOutput.log and CMakeError.log for clues

- If ***make*** fails…

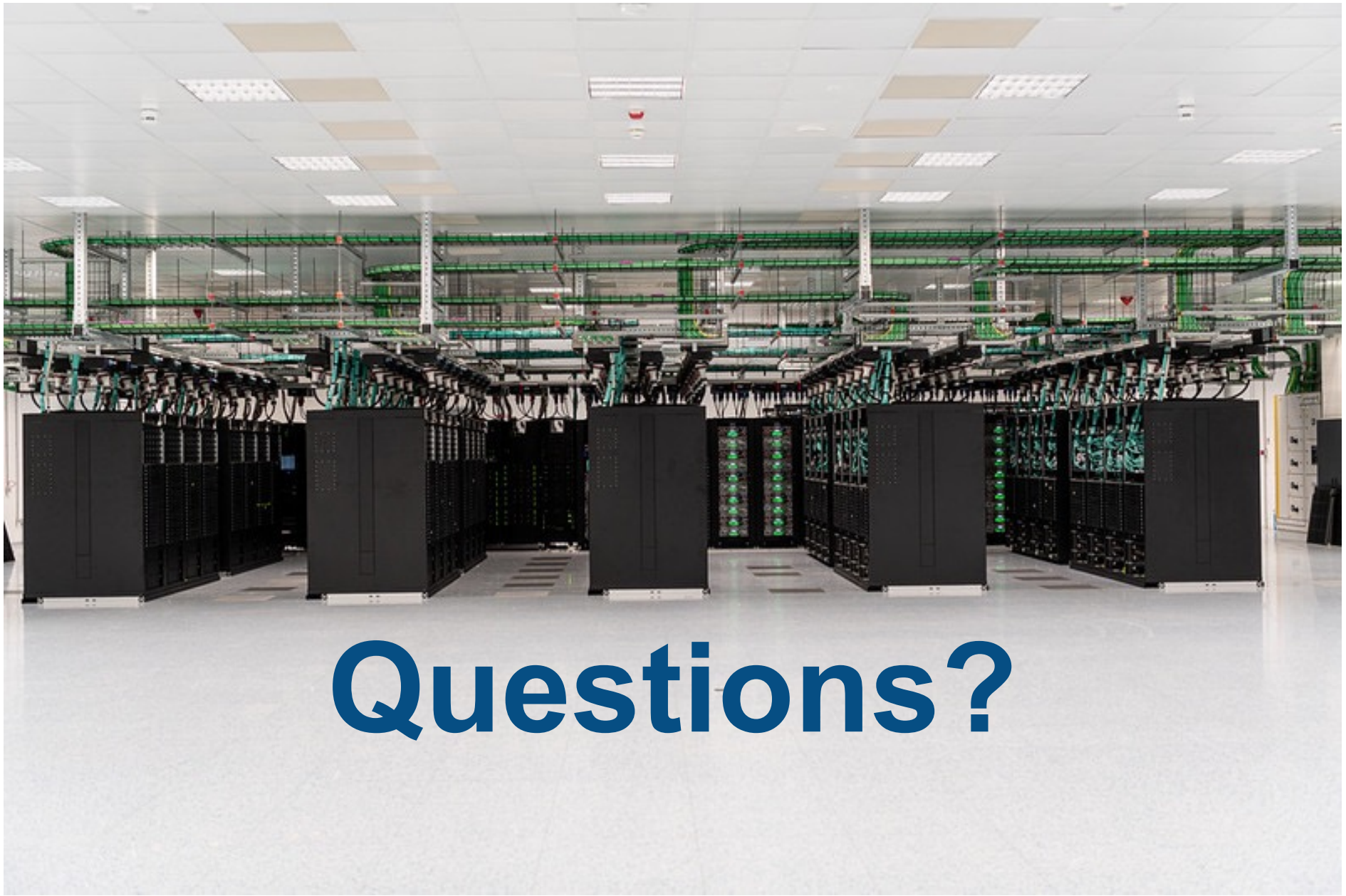
```
make VERBOSE=1
```

Step 3: Plain Makefiles / Custom builds

- You may need to edit Makefiles or included text files manually.
- Pay attention to relevant variables:
  - CC, FC, CXX, etc.
  - CFLAGS, FCFLAGS, CXXFLAGS, LDFLAGS , etc.
  - NETCDF_DIR, HDF5_DIR, or specific variables for dependencies.
- Same principles apply:
  - Check package documentation first!
  - Avoid hardcoding paths to dependencies, use variables from the relevant modules.
  - Don’t forget to define the rpath to dependent shared libraries in non-system locations.

The builder mindset

- Every software package is different, read the installation instructions first!
- Know your environment with modules.
- Never hardcode paths, use environment variables from modules.
- Use PERM (or TMPDIR for small cases) as a build directory. Avoid SCRATCH.
- Do not run big compilations interactively, use batch jobs.
- Avoid using LD_LIBRARY_PATH, use rpath instead at build time.
- Be tidy, do not mix compilers.



Questions?


ECMWF
Reception
Hands on time!
https://confluence.ecmwf.int/x/xwGaFQ
Xavi Abellan - ECMWF