Compiling software using cluster libraries

If you see the following error when compiling a code on the cluster:

```
fatal error: XXXX.h: No such file or directory
```

That means that the software you are trying to compile needs a specific header file provided by a third party library. In order to use a third party library, the compiler mainly needs two things:

- a header file XXXX.h
- the binary of the library: XXXXX.so

By default in Linux systems, those files are located in default paths as: /usr, /lib, etc.. There are two ways to tell the compiler where to look for those files: Makefile or using compiler variables.

Makefile

Makefiles provide the following Variables :

- CFLAGS
- CXXFLAGS
- FFLAGS
- LDFLAGS

The three first variables are used to pass extra options to a specific compiler and language, c, c++ and fortran respectively. The last variable is meant to be used to pass the option `-L -l` which are used by the linker.

Example

```
CFLAGS+= -I/usr/local/cuda/include
LDFLAGS+= -L/usr/local/cuda/lib -lcudnn
```

Here we will tell the compiler where to find the include files and the location of libraries. Those variables should already be present on the makefile and used on the compilation process.

GCC Variables

if you are using GCC, you can use the following Variables :

- CPATH
- LIBRARY_PATH

```
CPATH=/usr/local/cuda/include
LIBRARY_PATH=/usr/local/cuda/lib
```

This would have the same result as modifying the variable on the Makefile. This procedure is very useful in case you do not have access to the Makefile or Makefile variables are not used during compilation.

Using cluster libraries

On the cluster, libraries are provided by modules which means that you need to tell the compiler to look for headers files and binary files in special locations. The procedure is the following:

- load the library: module load XXX
- find the name of the ROOT variable by executing: module show XXX
- Use that variable on the CFLAFGS and LDFLAGS definition

Example

```
$ module load cuda
$ module show cuda
-------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
   /dcsrsoft/spack/arolle/v1.0/spack/share/spack/lmod/Zen2-IB/Core/cuda/11.6.2.lua:
-------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
whatis("Name : cuda")
whatis("Version : 11.6.2")
whatis("Target : zen")
whatis("Short description : CUDA is a parallel computing platform and programming model invented by NVIDIA.
It enables dramatic increases in computing performance by harnessing the power of the graphics processing
unit (GPU).")
help([[CUDA is a parallel computing platform and programming model invented by
NVIDIA. It enables dramatic increases in computing performance by
harnessing the power of the graphics processing unit (GPU). Note: This
package does not currently install the drivers necessary to run CUDA.
These will need to be installed manually. See:
```

```
https://docs.nvidia.com/cuda/ for details.]])
depends_on("libxml2/2.9.13")
prepend_path("LD_LIBRARY_PATH","/dcsrsoft/spack/arolle/v1.0/spack/opt/spack/linux-rhel8-zen/gcc-8.4.1/cuda-11.6.2-rswplbcorqlt6ywhcnbdisk6puje4ejf/lib64")
prepend_path("PATH","/dcsrsoft/spack/arolle/v1.0/spack/opt/spack/linux-rhel8-zen/gcc-8.4.1/cuda-11.6.2-rswplbcorqlt6ywhcnbdisk6puje4ejf/bin")
prepend_path("CMAKE_PREFIX_PATH","/dcsrsoft/spack/arolle/v1.0/spack/opt/spack/linux-rhel8-zen/gcc-8.4.1/cuda-11.6.2-rswplbcorqlt6ywhcnbdisk6puje4ejf/")
setenv("CUDA_HOME","/dcsrsoft/spack/arolle/v1.0/spack/opt/spack/linux-rhel8-zen/gcc-8.4.1/cuda-11.6.2-rswplbcorqlt6ywhcnbdisk6puje4ejf")
setenv("CUDA_ROOT","/dcsrsoft/spack/arolle/v1.0/spack/opt/spack/linux-rhel8-zen/gcc-8.4.1/cuda-11.6.2-rswplbcorqlt6ywhcnbdisk6puje4ejf")
```

You can observe that there is the variable `CUDA_ROOT` which is the one that should be used.

```
export CFLAGS="-I$CUDA_ROOT/include"
LDFLAGS+= -L$(CUDA_ROOT)/lib64/stubs -L$(CUDA_ROOT)/lib64/ -lcuda -lcudart -lcublas -lcurand
```

This is quite a complex example, sometimes you only need `-L$(XXX_ROOT)/lib`.

Example for R package

In the case of an R package, we do not have control over the Makefile, so the only option is to use GCC variables. For an R package that depend on gsl and mpfr libraries, we need to do the following:

```
mdoule load gsl mpfr
export CPATH=$GSL_ROOT/include:$MPFR_ROOT/include
export LIBRARY_PATH=$GSL_ROOT/lib:$MPFR_ROOT/lib
```

Révision #7
Créé 20 juillet 2023 08:32:57 par Cristian Ruiz
Mis à jour 20 juillet 2023 11:22:37 par Cristian Ruiz