NEMO5 Installation Guide

Table of Contents

Quick Start Guide

NEMO5 is only compatible with Linux platforms. To compile it you'll need the following:

- Some MPI implementation (compiled as a shared library)
- Python 2.7.x
- Boost 1.4.3 or greater.

NEMO5 depends on many third-party libraries that have to be manually compiled. Because of this NEMO5 is shipped with a `libs/` folder where the source and compilation script for each library are stored.

`To compile the third-party libraries go to the libs/ folder and run the configure script:`

```
cd libs;
export LIBS_TOP=$(pwd);
./configure --with-libs-top=$(LIBS_TOP)
```

To get help about the options of this configure script execute `./configure --help`. The `with-libs-top` option is just telling the configuration script where the `libs/` folder is.

This script will verify that all the required tools to compile the third-party libraries are available in your machine. If this fails you should send the generated log, available at `libs/config.log` to the NEMO5 development team through the support forum at https://nanohub.org/groups/nemo5distribution/forum/defaultsection/compilation.

If this script finishes successfully then you are ready to compile all the third-party libraries by executing `./build.sh`. This will try to build all the third party libraries by going inside each one of the libraries' folders, in the appropriate order, and executing `make`. If you have any problems during this step report the problem on the support group forum.

Once all the third-party libraries are ready go to prototype folder and execute:

```
cd prototype;
./configure --with-libs-top=$(LIBS_TOP)
make
```

Again, any problem can be reported at the forum. Once this is done you can just type `make` inside the `prototype` directory and wait until NEMO5 compilation finishes. If the compilation finishes correctly then you should see the binary executable file at `prototype/bin/nemo`.

To use NEMO5 please refer to the distribution and support page
https://nanohub.org/groups/nemo5distribution

Introduction

There are currently two methods for building NEMO5, the *manual configure system* and the *automatic configure system*. New users should follow the *automatic configure system*. The manual configure system is documented here for completeness.

Users should be familiar with Linux, compiling and linking, and need to be familiar with the software already available on their machine and how to link to it, as they will have to compile many of the required third-party libraries.

The manual configure system is more robust, but requires definitions of many specific variables to the infrastructure. Additionally the dependencies and the errors that this system shows unfortunately don't point you to anywhere unless you have experience with the error messages. The automatic configure system has been developed which aims to guide users in the right direction about missing dependencies. This system outputs a `config.log` file which can help with diagnosing build problems.

Where to get help
If at some point it is needed the user can ask for help at the NEMO5 support forum : https://nanohub.org/groups/nemo5distribution/forum

SVN
For close collaborators, you may have SVN access to NEMO5. The repository and associated checkout command is shown below. The username and password are the same as your nanoHUB.org credentials.

```
svn checkout https://nanohub.org/tools/nemo/svn/trunk/NEMO .
```

Directory structure:

- / : NEMO5's root directory
    - prototype/: NEMO 5 source code
        - manual/: user manual (compile using latex ../bin/nemo --manual)
        - doc/: doxygen documentation (developer manual)
        - include/: .h-files, needed e.g. for using NEMO 5 as a library
        - src/: .cpp-files
        - bin/: location of nemo executable
    - libs/: 3rd party libraries

Compiling NEMO5

NEMO5 dependencies

NEMO5 is distributed with all its required third-party libraries sources, because some of them require special compilation parameters to link correctly against NEMO5's code. These sources can be found on the `libs` directory under the *root NEMO5 directory*. Each folder inside the `libs` directory contains at least two things: (1) a `Makefile` file and (2) a source code folder or compressed source code folder file.
The third-party libraries dependencies of NEMO5, and their corresponding dependencies, are depicted in Figure 1.

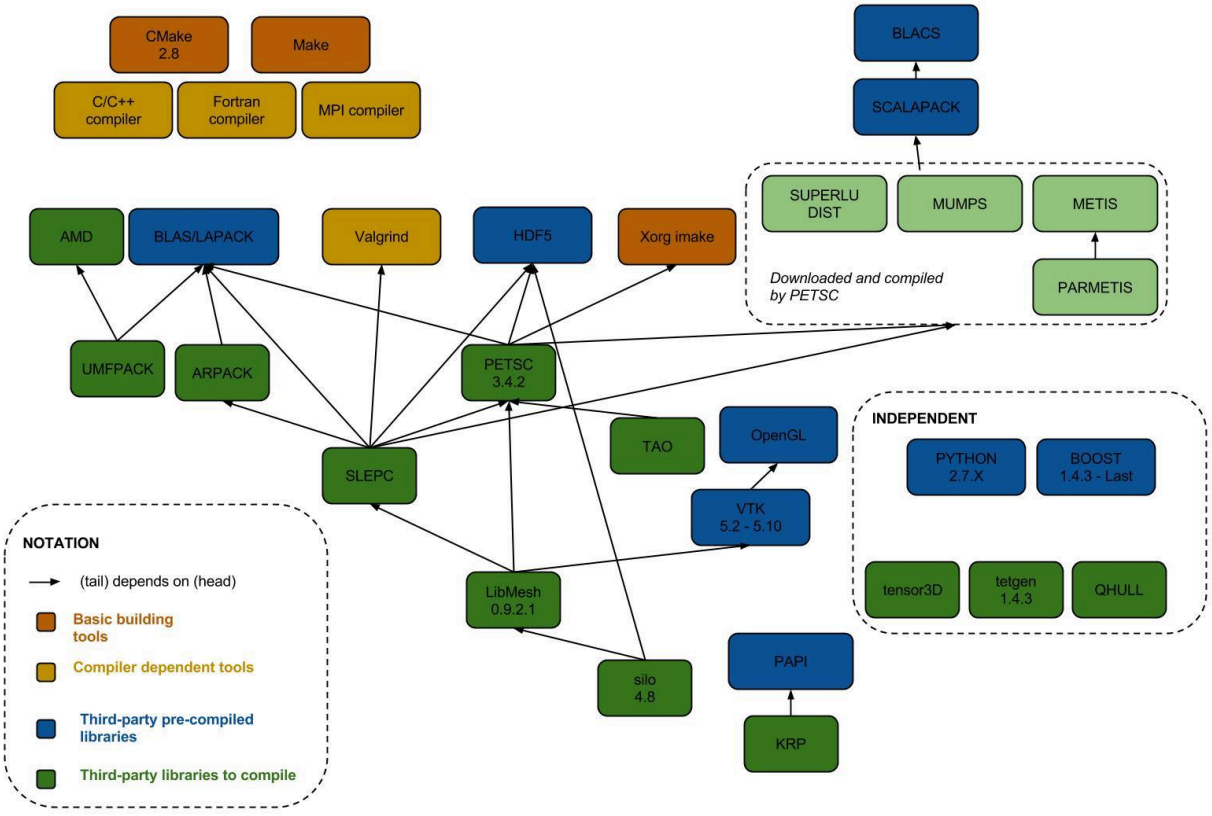

Figure 1: NEMO5 dependencies

Compiling NEMO5 dependencies

Before starting make sure that your environment has all the brown and gold tools from the Figure 1 installed (Valgrind is the only optional one) and Python >= 2.7 but less than 3.

Ubuntu

On Ubuntu this is done by executing:

```
sudo apt-get update

sudo apt-get install build-essential openmpi-bin openmpi-common
libopenmpi-dev libopenmpi1.6 libhdf5-openmpi-dev gfortran liblapack-dev
libboost-dev libvtk5.8 libvtk5-dev libhdf5-mpi-dev hdf5-tools h5utils cmake
python-all-dev libboost-all-dev
```

Configuration methods

To compile any of these libraries you'll first need to **configure the required libraries' paths** and then go inside each library's folder and **run** `make`. The way to go about each of these steps two will depend on the compilation method that you select. The two supported methods are referred to as : *the automatic configure system* and *the manual configure system*.

Note that NEMO5 uses a unique installation of PETSc and SLEPc which uses complex and real builds simultaneously. Although PETSc and SLEPc may be already available on your machine, those builds will not work. You must build it from source. Additionally, PETSc requires HDF5 and NEMO5 requires HDF5. Even though HDF5 modules are usually available on some systems, we generally link to PETSc's HDF5.

It is recommended to use appropriate numerical packages if possible: BLAS, LAPACK, MKL. If this is your first NEMO5 build, it is possible to skip these until you get a functioning executable, then go back and work on performance-related issues.

Compiling NEMO5

Once all the required third-party libraries are available you can go on to compile NEMO5 by executing the `make` command inside the `prototype/` directory. Keep in mind the following options for this `make` command:

- `make METHOD=opt` : Will compile NEMO5 in optimized mode, deactivating some profiling and debugging utilities. This is the default behaviour of the make command if no `METHOD` is specified.

- `make METHOD=dbg` : Will compile NEMO5 with the `-g` flag for compatibility with `gdb` and will also compile some specific debugging tools inside NEMO5's code, e.g. memory leaks detection utilities.

Automatic configure system

Compiling third-party libraries

To ***configure the required environment*** go to the *NEMO5's root directory* and then to the `libs/` directory and run `./newconfigure.sh`. This will try to automatically detect all the required information to compile all the libraries, just like any other autotools configuration file. If this command runs successfully then you should see something like

```
configure: creating ./config.status
config.status: creating petsc/make.inc
config.status: creating petsc/Makefile
config.status: creating slepc/make.inc
config.status: creating slepc/Makefile
config.status: creating ARPACK/make.inc
config.status: creating ARPACK/Makefile
config.status: creating ARPACK/PUTIL/Makefile
config.status: creating ARPACK/UTIL/Makefile
config.status: creating ARPACK/PSRC/Makefile
config.status: creating ARPACK/SRC/Makefile
config.status: creating libmesh/make.inc
config.status: creating libmesh/Makefile
config.status: creating silo/make.inc
config.status: creating silo/Makefile
config.status: creating AMD/make.inc
config.status: creating AMD/Makefile
config.status: creating AMD/Source/Makefile
config.status: creating UMFPACK/make.inc
config.status: creating UMFPACK/Makefile
config.status: creating UMFPACK/Source/Makefile
config.status: creating QHULL/make.inc
config.status: creating QHULL/Makefile
config.status: creating QHULL/src/Makefile
config.status: creating qhull-2010.1/make.inc
config.status: creating qhull-2010.1/Makefile
config.status: creating qhull-2010.1/src/Makefile
```

```
config.status: creating qhull-2010.1/cpp/Makefile
config.status: creating tetgen1.4.3/make.inc
config.status: creating tetgen1.4.3/Makefile
config.status: creating tensor3D/make.inc
config.status: creating tensor3D/Makefile
config.status: creating TAO/make.inc
config.status: creating TAO/Makefile
```

And later something like:

```
configure: creating ./config.status
config.status: creating make.inc
config.status: creating src/math/libnemo_petsc_double/Makefile
config.status: creating src/math/libnemo_petsc_complex/Makefile
```

A `make.inc` (not a `make.inc.in` !) file should've been generated inside each subfolder of the `libs/` folder. If this ran successfully then you can compile the libraries by executing:

```
cd libs/
./build.sh
```

What's this configure script doing?

The `./newconfigure.sh` is just doing the executing the following behind the scenes, when no additional parameter is supplied:

```
cd libs;
./configure
cd prototype;
./configure
```

If something goes wrong with this command you can manually execute the above commands to troubleshoot the configuration process. Keep in mind that by executing the configure commands you can specify the path to specific libraries like VTK, PETSc or SLEPc. An example of this can be seen at `conf/archlinux/configure.sh`. More details are available when you run `./configure –help`

Each library subfolder under the `libs/` directory (e.g. petsc, slepc, silo) has a `make.inc.in` file

which will be converted in a `make.inc` file by the configure script, once all the libraries' directories and the paths to the compilation tools are detected. This variables store library paths, compilation tool paths, libraries versions, etc … that are needed for the "make" command to work on each subdirectory of libs/.

If for any reason the configure script fails you can manually define the variables needed by each library by replacing the every name surrounded by at signs (@) by the relevant value, e.g. `@MPICC@` should be replaced by `mpicc` or your mpi compiler of choice.

If any of this commands fail it'll show you the reason why it failed and it'll also store all the relevant trace of its execution on a `config.log` file under the `libs/` folder and under the `prototype/` folder. You can submit this file to the NEMO5's development team to get help. Please keep in mind that this configuration tool assumes that VTK 5.10 is installed.

Troubleshooting

- We know of issues when statically linking NEMO5 to MPICH3.1. We had this problem at TianHe-2 compiling everything from scratch and using the following stack:
- Python usually will need UCS4 support to link against Boost.Python and NEMO5 correctly. This is solved by compiling Python with the flags --enable-unicode=ucs4 --enable-shared and when Boost is compiled, after ./bootstrap.sh, modify the file project-config.jam to point to the compiled version of Python as using
- using python
  : 2.7
  : python : 2.7 : /work/02606/srubiano/NEMO4/opt
  : /work/02606/srubiano/NEMO4/opt/include
  : /work/02606/srubiano/NEMO4/opt/lib
  : <toolset>intel
  ;
-
-

Specific architectures

Stampede (USING MPICH 3.1):

IMPORTANT! Make sure to have only the Intel MPI module loaded when compiling, otherwise this could lead to MPI symbols crashes/incompatibilities.

IMPORTANT! Build in the $WORK directory (see below)

```
module purge;
module load intel/14.0.1.106
module load hdf5/1.8.12
module load cmake/2.8.9
module load impi/4.1.0.030

cd $WORK
mkdir NEMO;
INSTALL_DIR=$(pwd)
cd $INSTALL_DIR/NEMO/;
svn co https://nanohub.org/tools/nemo/svn/trunk/NEMO .
cd $INSTALL_DIR/NEMO/libs/vtk/;
make -f Makefile.stampede;
cd $INSTALL_DIR/NEMO/libs/python/;
make -f Makefile.stampede INSTALL_PATH=$INSTALL_DIR;
cd $INSTALL_DIR/NEMO/libs/boost/; tar xfz boost_1_51_0.tar.gz;
cd  boost_1_51_0/ ;
./bootstrap.sh --prefix=$INSTALL_DIR --libdir=$INSTALL_DIR/lib/;
```

Run the command

```
echo $INSTALL_DIR
```

This will return a directory, like `/work/02408/username` which you will use in the next step as your INSTALL_DIR.

Use an editor to open project-config.jam and replace the line that looks like:

```
using python : 2.6 : /usr ;
```

with:

```
using python
     : 2.7
     : <YOUR INSTALL_DIR HERE>/bin/python
     : <YOUR INSTALL_DIR HERE>/include/python2.7
     : <YOUR INSTALL_DIR HERE>/lib/python2.7
     : <toolset>intel
     ;
```

Make sure to replace the " <YOUR INSTALL_DIR HERE>" by the directory stored in the bash variable INSTALL_DIR. After this it will look like:

```
using python
     : 2.7
     : /work/02408/username/bin/python
     : /work/02408/username/include/python2.7
     : /work/02408/username/lib/python2.7
     : <toolset>intel
     ;
```

Then execute

```
./bjam install -j 16 toolset=intel -a
```

At the end of this the message

```
"...updated 10696 targets..."
```

should appear, there should **NO** message like :

```
...failed updating 56 targets...
```

Then do:

```
cd $INSTALL_DIR/NEMO/libs/
```

```
export LD_LIBRARY_PATH=$INSTALL_DIR/lib/:$LD_LIBRARY_PATH

./configure --with-lapack=" -openmp -mkl=parallel
-offload-attribute-target=mic  -lpthread -lm" --with-blas=" -openmp
-mkl=parallel   -offload-attribute-target=mic  -lpthread -lm -L$ICC_LIB"
--with-vtk=$INSTALL_DIR/NEMO/libs/vtk --with-hdf5=$TACC_HDF5_BIN/h5cc
PYTHON=$INSTALL_DIR/bin/python --with-cmake=/opt/apps/cmake/2.8.9/bin/
CC=icc CXX=icpc  MPICC=mpicc MPICXX=mpic++
```

If this finishes correctly execute the commands in libs/build.sh by hand making sure that no evident error is shown. Once all the commands in the build.sh script have been executed one can move on to compile NEMO5 at the prototype directory:

```
ln -s $INSTALL_DIR/NEMO/libs/libmesh/libmesh-0.9.3/
$INSTALL_DIR)/NEMO/libs/libmesh/libmesh


export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$INSTALL_DIR/lib

cd $INSTALL_DIR/NEMO/prototype;
./configure --with-lapack=" -openmp -mkl=parallel
-offload-attribute-target=mic  -lpthread -lm" --with-blas=" -openmp
-mkl=parallel   -offload-attribute-target=mic  -lpthread -lm -L$ICC_LIB"
--with-vtk=$INSTALL_DIR/NEMO/libs/vtk --with-hdf5=$TACC_HDF5_BIN/h5cc
PYTHON=$INSTALL_DIR/bin/python --with-cmake=/opt/apps/cmake/2.8.9/bin/
CC=mpiicc CXX=mpiicpc  MPICC=mpiicc MPICXX=mpiicpc
--with-boost=$INSTALL_DIR --with-libs-top=$INSTALL_DIR/NEMO/libs
```

Open the generated make.inc file and add `-DMPICH_SKIP_MPICXX` to the CXXFLAGS. Remove the pound in front of the `MKLHOME` line.

To compile for MICs:

Go to src/math/RGF_library/Makefile and add the following CXXFLAGS :
-offload-attribute-target=mic

To run NEMO5 on MICs remember to first execute:

```
PCLMICCORES=`micinfo | grep "Total No of Active Cores" | awk '{print
$7}' | head -n 1`
PCLMICCORES=$((PCLMICCORES-1))
PCLMICTHREADS=$((PCLMICCORES*4))
echo ""
echo "Detected" $PCLMICTHREADS "threads for Xeon Phi offloads"


export OMP_NUM_THREADS=8
export KMP_AFFINITY=proclist=[0-15],granularity=thread,explicit
export MIC_ENV_PREFIX=MIC
export MIC_OMP_NUM_THREADS=$PCLMICTHREADS
export MIC_KMP_AFFINITY=compact
export MIC_USE_2MB_BUFFERS=64K
```

Manual Configuration

To ***configure the required environment*** go to the *root NEMO5 directory* and run `./configure.sh <configuration id>`. For the list of available platforms and compilers execute configure.sh without arguments. Most of these are Purdue specific machines. If you are building a new platform, you should start with one and edit it for your system. There is platform “custom” which can be used for this purpose.

The very basic steps to build NEMO5 are to run the configure.sh script, build the libraries, and compile NEMO5. Configuration parameters and other build parameters (e.g. compilers, some library version numbers) are stored in several places (section II). Once those parameters are properly set, run the configuration script (section I) and then build the libraries (section III), before building NEMO5 (section IV).

I. Execute the configuration script configure.sh with
a platform-dependent argument:

e.g. INTEL compiler:
./configure.sh custom

II. The configure.sh script will read variables from the <NEMO5>/mkfiles/make.inc.yourplatform (e.g. make.inc.custom) file.

Other files that should not have to be edited, but listed here for completeness are <NEMO5>/make.inc.template, <NEMO5>/prototype/make.common.template, <NEMO5>/prototype/Make.rules, and <NEMO5>/prototype/Makefile. When you run configure.sh, <NEMO5>/make.inc is created from <NEMO5>/make.inc.template and <NEMO5>/prototype/Make.common is created from make.common.template.

III. The necessary libraries are in <NEMO5>/libs directory or you can link to them if they exist elsewhere on your system. Each directory will have a Makefile in it. Usually the actually "Makefile" has appropriate build parameters. For PETSc and libMesh, use the Makefile.custom.

1. VTK: anything between 5.2 to 5.10. Not 6.x.
2. BOOST (1.43+)
3. PYTHON (2.7.2+ and less than 3)
4. PETSc 3.4.3.
5. SLEPc compatible with PETSc (generally version number of SLEPc is same as PETSc's)
6. libMesh 0.9.2.1
7. AMD, ARPACK (SLEPc relies on this), qhull-2010.1, UMFPACK, tensor3D, tetgen1.4.3, silo
8. TAO 2.2-pre1(optional for PETSc 3.4.x builds; requires PETSC 3.4.x; not used for older PETSc builds)

9. HDF5, generally you can link to PETSc's build of this. It should be located in linux/externalpackages/hdf5-1.8.10-patch1
10. For GPUs-MAGMA (developmental)
11. These libraries may have other common dependencies.

The order in which the packages are built does not matter except:
1. SLEPc relies on PETSc
2. SLEPc (optionally) relies on ARPACK
3. libMesh relies on PETSc and SLEPc
4. TAO relies on PETSc

NOTES

- The PETSc Makefile.custom has configure options if MKL is installed and if it is not. Since we build real and complex versions of PETSc, there will be two configure lines.
- You can ignore an error about szlib if it occurs when building silo.
- You can ignore an error about “machine.aux” when compiling NEMO5 in prototype. This should only have during the first build.

IV. Building NEMO5

```
cd <...>/NEMO; ./configure.sh custom

cd libs/ARPACK; make

cd ../petsc; make -f Makefile.custom

cd ../slepc; make

cd ../AMD; make

cd ../qhull-2010.1; make

cd ../UMFPACK; make

cd ../tensor3D; make

cd ../tetgen1.4.3; make

cd ../libmesh; make -f Makefile.custom

cd ../silo; make

cd ../../prototype; make -j8
```

You should have a ‘nemo’ binary in the prototype/bin directory.

Compile Doxygen Documentation

cd prototype/doc; make

Compile Manual

nemo binary must exist:

cd prototype/manual; ../bin/nemo --manual

Command Line Help

./nemo --help

Memory and Timing Profiling

NEMO5 has built in profiling
./nemo input.in --profiling // generate a xml file with all information possible
./nemo input.in --profiling memory // generate a xml with only system memory
./nemo input.in --profiling time // generate a xml with only time