J Mak NEMO notes

Julian Mak

Oct 25, 2023

CONTENTS:

Homebrew collection of NEMO ($\geq$ v3.6) related content, existing here primarily to remind myself of code details. Includes compilation notes, analyses codes, and misc. other topics. Appropriate disclaimers in the individual sections themselves.

I can be contacted at jclmak@ust.hk regarding modifying / adding to the content here.

- If you are here for the GEOMETRIC codes that I maintain, try this repository.

CHAPTER

ONE

NEMO COMPILATION NOTES

These are just my own notes for compiling NEMO on a variety of clusters and computers in a public place largely so I can look it up as long as I have internet; if it happens useful for you, great! Please consult the NEMO page for the official details.

While it is fairly straightforward on a supported cluster/supercomputer (e.g. try NOCL ARCHER guide) it can be a bit temperamental on a local machine largely down to library and compiler compatibility. The following notes are what I did to get XIOS and NEMO compiling and running, and will display commands with `gcc4.9` compilers (which is my default for other reasons). Extra things that need to be modified for other compilers I have tested will be given accordingly (see the top of the individual pages as to which compilers I have tested the notes with).

I added the following to my `~/.bashrc` so as to override the default compilers I had (change these if need be):

```
export CC=/usr/bin/gcc-4.9
export CXX=/usr/bin/g++-4.9
export FC=/usr/bin/gfortran-4.9
export F77=/usr/bin/gfortran-4.9
export CPP=/usr/bin/cpp-4.9
```

1.1 NEMO 3.6 (stable) + XIOS 1.0

Tested with

- `gcc4.9`, `gcc5.4` on a linux system
- `gcc4.8` on a Mac (El Capitan OSX 10.11)

The assumption here is that the compiler is fixed and the packages (e.g., NetCDF4 and a MPI bindings) are configured to be consistent with the compilers. See *here* to check whether the binaries exist, where they are, and how they might be installed separately if need be. All the `#CHANGE ME` highlighted below needs be modified to point to the appropriate paths or binaries (soft links with `ln -s` are ok).

The instructions below uses `gcc4.9` for demonstration (modifications with `gcc5.4` as appropriate). I defined some extra variables on a Linux machine:

```
export $BD=/home/julian/testing/gcc4.9-builds # CHANGE ME

export C_INCLUDE_PATH=$BD/install/include:$C_INCLUDE_PATH
export CPLUS_INCLUDE_PATH=$BD/install/include:$CPLUS_INCLUDE_PATH
export LIBRARY_PATH=$BD/install/lib:$LIBRARY_PATH
export LD_LIBRARY_PATH=$BD/install/lib:$LD_LIBRARY_PATH
```

You shouldn't need to do the above if the packages are forced to look at the right place (e.g. via `-L` and/or `-I` flags with path to libraries and include files respectively). Not all of these are necessary depending on whether you choose to build/have static or dynamic libraries, and the `LD_LIBRARY_PATH` seems to sort out a lot of problems with linking libraries.

On a Mac done through anaconda the above was not necessary. My understanding is that setting these variables might not actually do anything unless an option is specifically enabled in Xcode.

1.1.1 XIOS 1.0 (svn v703)

To use NEMO you probably do need XIOS to do the I/O. The instructions here follow the one given in the XIOS instructions with any errors that arise. A useful site to search for XIOS related errors may be found on the XIOS user mailing list.

Here XIOS1.0 is used with NEMO3.6 for compatibility reasons. For the purposes here I created a folder called `XIOS` and used `svn` to get XIOS1.0 (which is going to be `XIOS/xios1.0`):

```
mkdir XIOS
cd XIOS
svn checkout -r 703 http://forge.ipsl.jussieu.fr/ioserver/svn/XIOS/branchs/xios-1.0 xios-
↪1.0
```

To get XIOS to compile, the compilers and packages need to be pointed to first, via modifying files in `arch`. Since I am using `gcc`, I did the following just to make a fresh copy:

```
cd xios1.0/arch
cp arch-GCC_LINUX.env arch-GCC_local.env
cp arch-GCC_LINUX.fcm arch-GCC_local.fcm
cp arch-GCC_LINUX.path arch-GCC_local.path
```

The `*.env` file specifies where HDF5 and NetCDF4 libraries live. The `*.fcm` file specifies which compilers and options to use. The `*.path` file specifies which paths and options to include. My files look like the following:

```
# arch-GCC_local.env

export HDF5_INC_DIR=/usr/local/include       # CHANGE ME
export HDF5_LIB_DIR=/usr/local/lib           # CHANGE ME

export NETCDF_INC_DIR=/usr/local/include     # CHANGE ME
export NETCDF_LIB_DIR=/usr/local/lib         # CHANGE ME
```

You could get an idea where the HDF5 and NetCDF4 directories are by doing `which h5copy` and `which nc-config` (assuming these are on `$PATH`), which should give you a `directory/bin`, and it is the `directory` part you want. If you did install the libraries somewhere else as in *other packages*, say, then make sure the `which` commands are pointing to the right place.

```
# arch-GCC_local.fcm

################################################################################
###################                Projet XIOS               ###################
################################################################################

%CCOMPILER      /usr/local/bin/mpicc               # CHANGE ME
%FCOMPILER      /usr/local/bin/mpif90              # CHANGE ME
```

(continues on next page)

(continued from previous page)

```
%LINKER         /usr/local/bin/mpif90               # CHANGE ME

%BASE_CFLAGS    -ansi -w
%PROD_CFLAGS    -O3 -DBOOST_DISABLE_ASSERTS
%DEV_CFLAGS     -g -O2
%DEBUG_CFLAGS   -g

%BASE_FFLAGS    -D__NONE__
%PROD_FFLAGS    -O3
%DEV_FFLAGS     -g -O2
%DEBUG_FFLAGS   -g

%BASE_INC       -D__NONE__
%BASE_LD        -lstdc++

%CPP            cpp-4.9                             # CHANGE ME
%FPP            cpp-4.9 -P                          # CHANGE ME
%MAKE           make
```

Check the MPI locations and versions by doing `which mpicc` and `mpicc --version` say. If they are the right ones you could just have `mpicc` instead of the full path as given above. MPI bindings are used here to avoid a possible error that may pop up in relation to the build trying to find `mpi.h`. The `gmake` command was swapped out by the `make` command (I don't have `cmake` on the laptop).

Note: For `gcc5.4` and maybe newer versions, doing just the above when compiling leads to a whole load of errors about clashing in C++:

```
.../include/boost/functional/hash/extensions.hpp:69:33: error: 'template<class T, class␣
↪A> std::size_t boost::hash_value' conflicts with a previous declaration
 std::size_t hash_value(std::list<T, A> const& v)
                                ^
```

Adding `-D_GLIBCXX_USE_CXX11_ABI=0` to `%BASE_CFLAGS` fixes these.

```
# arch-GCC_local.path

NETCDF_INCDIR="-I$NETCDF_INC_DIR"
NETCDF_LIBDIR="-Wl,'--allow-multiple-definition' -L$NETCDF_LIB_DIR"
NETCDF_LIB="-lnetcdff -lnetcdf"

MPI_INCDIR=""
MPI_LIBDIR=""
MPI_LIB=""

HDF5_INCDIR="-I$HDF5_INC_DIR"
HDF5_LIBDIR="-L$HDF5_LIB_DIR"
HDF5_LIB="-lhdf5_hl -lhdf5 -lhdf5 -lz"
```

The above has all the OASIS (the atmosphere / ocean coupler) keys removed. I added the `-Wl, '--allow-multiple-definition'` key for reasons I don't remember anymore...

Now it should be ready to compile. Assuming the current directory is `xios1.0/arch`:

```
cd ../
./make_xios --full --prod --arch GCC_local -j2 |& tee compile_log.txt
```

The `-j2` option uses two processors to build. The `tee` command is to keep logs of potential errors (the `|&` is short for `2>&1 |`) for debugging errors that may arise.

Note: If you get something like

```
/home/julian/testing/nemo-6800/xios-703/xios-1.0/inc/netcdf.hpp:20:26: fatal error:␣
↪netcdf_par.h: No such file or directory
 #  include <netcdf_par.h>
                          ^
compilation terminated.
fcm_internal compile failed (256)
/home/julian/testing/nemo-6800/xios-703/xios-1.0/Makefile:1620: recipe for target
↪'inetcdf4.o' failed
```

then it is probably because NetCDF4 was not built as parallel. There is a actually a copy of the file in `./extern/src_netcdf4/netcdf_par.h`, and it could be pointed to by looking into `bld.cfg`:

```
bld::tool::cflags    %CFLAGS %CBASE_INC -I${PWD}/extern/src_netcdf -I${PWD}/extern/boost/
↪include -I${PWD}/extern/rapidxml/include -I${PWD}/extern/blitz/include
```

where `src_netcdf` should be changed to `src_netcdf4`.

Note: If you get something like

```
libhdf5.a(H5PL.o): undefined reference to symbol 'dlclose@@GLIBC_2.2.5'
```

then this suggests that the HDF5 library that is being called is built as a static and/or not shareable library. In this case adding the `-ldl` flag to `HDF5_LIB` in `arch-GCC_local.path` should work. Or if you want to you can recompile HDF5 as a shareable library; see *other packages* on how you might go about doing this.

It should work and takes around 5 mins to compile for me. The main end result is are binaries in `xios1.0/bin/` which NEMO will call.

Note: Do `ldd bin/xios_server.exe` (or wherever `xios_server.exe` lives) to make sure the libraries linked to it are the intended libraries. XIOS may still work if the NetCDF versions are ok, but if not, go back and define `LD_LIBRARY_PATH` and other variables accordingly; see above.

`xios_server.exe` is one of the other binaries built from compiling but is not required for small runs on a laptop. For its use on a cluster see for example the instructions on the NOCL ARCHER guide.

1.1.2 NEMO 3.6 (svn v6800)

Check out a version of NEMO. I have another folder separate to the XIOS folders to contain the NEMO codes and binaries:

```
mkdir NEMO
cd NEMO
svn checkout -r 6800 http://forge.ipsl.jussieu.fr/nemo/svn/NEMO/trunk nemo3.6-6800
```

This checks out version 6800 (NEMO 3.6) and dumps it into a folder called `nemo3.6-6800` (change the target path to whatever you like).

Note: `svn checkout https://forge.ipsl.jussieu.fr/nemo/svn/NEMO/releases/release-3.6 nemo3.6` would pull the official version

A similar procedure to specify compilers and where XIOS lives needs to be done for NEMO. Again, because of the compilers I am using:

```
cd nemo3.6-6800/NEMOGCM/ARCH
cp OLD/arch-gfortran_linux.fcm ./arch-gfortran_local.fcm
```

None of the fcm files associated with gfortran actually worked for me out of the box so here is my build of it (click HERE for a detailed log of how I got to the following):

```
# gfortran_local.fcm

# generic gfortran compiler options for linux
# NCDF_INC    netcdf include file
# NCDF_LIB    netcdf library
# FC          Fortran compiler command
# FCFLAGS     Fortran compiler flags
# FFLAGS      Fortran 77 compiler flags
# LD          linker
# LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
# FPPFLAGS    pre-processing flags
# AR          assembler
# ARFLAGS     assembler flags
# MK          make
# USER_INC    additional include files for the compiler,  e.g. -I<include dir>
# USER_LIB    additional libraries to pass to the linker, e.g. -l<library>

%NCDF_HOME           /usr/local                                          # CHANGE ME

%XIOS_HOME           /home/julian/testing/gcc4.9-builds/XIOS/xios-1.0 # CHANGE ME

%CPP                 cpp-4.9                                             # CHANGE ME
%CPPFLAGS            -P -traditional

%XIOS_INC            -I%XIOS_HOME/inc
%XIOS_LIB            -L%XIOS_HOME/lib -lxios

%NCDF_INC            -I%NCDF_HOME/include
%NCDF_LIB            -L%NCDF_HOME/lib -lnetcdf -lnetcdff -lstdc++
```

(continues on next page)

(continued from previous page)

```
%FC                   mpif90                                                  # CHANGE ME
%FCFLAGS              -fdefault-real-8 -O3 -funroll-all-loops -fcray-pointer -cpp -ffree-
↪line-length-none
%FFLAGS               %FCFLAGS
%LD                   %FC
%LDFLAGS
%FPPFLAGS             -P -C -traditional
%AR                   ar
%ARFLAGS              -rs
%MK                   make
%USER_INC             %XIOS_INC %NCDF_INC
%USER_LIB             %XIOS_LIB %NCDF_LIB
```

The main changes are (see here for an attempt at the reasoning and a log of errors that motivates the changes):

- added `%NCDF_HOME` to point to where NetCDF lives
- added `%XIOS_*` keys to point to where XIOS lives
- added `%CPP` and flags, consistent with using `gcc4.9`
- added the `-lnetcdff` and `-lstdc++` flags to NetCDF flags
- using `mpif90` which is a MPI binding of `gfortran-4.9`
- added `-cpp` and `-ffree-line-length-none` to Fortran flags
- swapped out `gmake` with `make`

Note: It might be worthwhile doing the following first:

```
cd ../CONFIG/
./makenemo -j0 -r GYRE -n GYRE_testing -m gfortran_local
```

Then, add `key_nosignedzero` to the end of `/GYRE_testing/cpp_GYRE_testing.fcm` (see note at the bottom of the page). `-j0` does all the folder creation and copying but not the compile step.

To compile a configuration (using the GYRE config):

```
cd ../CONFIG/
./makenemo -j2 -r GYRE -n GYRE_testing -m gfortran_local |& tee compile_log.txt
```

This uses two processors, with `GYRE` as a reference, builds a new folder called `GYRE_testing`, with the specified architecture file, and outputs a log.

Note: The `-r GYRE` flag here only needs to be done once to create an extra folder and add `GYRE_testing` to `cfg.txt`. The subsequent compilations should then read, e.g., `./makenemo -n GYRE_testing -m gfortran_local`.

Check that it does run with the following:

```
cd GYRE_testing/EXP00
mpiexec -n 1 ./opa
```

This may be `mpirun` instead of `mpiexec`, and `-n 1` just runs it as a single core process. Change `nn_itend = 4320` in `nn_itend = 120` to only run it for 10 days (`rdt = 7200` which is 2 hours). With all the defaults as is, there should

be some GYRE_5d_*.nc data in the folder. You can read this with ncview (see the ncview page or, if you have sudo access, you can install it through sudo apt-get install ncview), bearing in mind that this is actually a rotated gyre configuration (see the following NEMO forge page or search for gyre in the NEMO book).

Note: My run actually crashed immediately. Looking into ocean.output and searching for E R R O R shows that key_nosignedzero needed to be added to /GYRE_testing/cpp_GYRE_testing.fcm. Rebuilding with the key then works fine.

Note: If your installation compiles but does not run with the following error

```
dyld: Library not loaded: @rpath/libnetcdff.6.dylib
Referenced from: /paths/./nemo
Reason: no suitable image found.  Did find:
/usr/local/lib/libnetcdff.6.dylib: stat() failed with errno=13
```

then it is not finding the right libraries. These could be fixed by adding the -Wl,-rpath,/fill me in/lib flag to the relevant flags bit in the *.fcm files (or possibly in XIOS the path and/or env) to specify exactly where the libraries live. This can happen for example on a Mac or if the libraries are installed not at the usual place.

Note: One infuriating problem I had specifically with a Mac (though it might be a gcc4.8 issue) is that the run does not get beyond the initialisation stage. Going into ocean.output and searching for E R R O R shows that it complained about a misspelled namelist item (in my case it was in the namberg namelist). If you go into output.namelist.dyn and look for the offending namelist is that it might be reading in nonsense. This may happen if the comment character ! is right next to a variable, e.g.

```
ln_icebergs = .true.!this is a comment
```

Fix this by adding a white space, i.e.

```
ln_icebergs = .true. !this is a comment
```

1.2 NEMO 3.7/4.0 + XIOS 2.0

Tested with

- gcc4.9, gcc5.4 on a linux system
- gcc4.8 on a Mac (El Capitan OSX 10.11)

This is the version I first implemented GEOMETRIC in, which is a development version I guess (?) that eventually led to NEMO 4.0. The code structure largely follows NEMO 3.6 but the commands are slightly different.

If you get errors that are not documented here, see if *the XIOS1.0 NEMO3.6* page contain the relevant errors.

The assumption here is that the compiler is fixed and the packages (e.g., NetCDF4 and a MPI bindings) are configured to be consistent with the compilers. See *here* to check whether the binaries exist, where they are, and how they might be installed separately if need be. All the #CHANGE ME highlighted below needs be modified to point to the appropriate paths or binaries (soft links with ln -s are ok).

The instructions below uses gcc4.9 for demonstration (modifications with gcc5.4 as appropriate). I defined some extra variables on a Linux machine:

```
export $BD=/home/julian/testing/gcc4.9-builds # CHANGE ME

export C_INCLUDE_PATH=$BD/install/include:$C_INCLUDE_PATH
export CPLUS_INCLUDE_PATH=$BD/install/include:$CPLUS_INCLUDE_PATH
export LIBRARY_PATH=$BD/install/lib:$LIBRARY_PATH
export LD_LIBRARY_PATH=$BD/install/lib:$LD_LIBRARY_PATH
```

You shouldn't need to do the above if the packages are forced to look at the right place (e.g. via `-L` and/or `-I` flags with path to libraries and include files respectively). Not all of these are necessary depending on whether you choose to build/have static or dynamic libraries, and the `LD_LIBRARY_PATH` seems to sort out a lot of problems with linking libraries.

On a Mac done through anaconda the above was not necessary. My understanding is that setting these variables might not actually do anything unless an option is specifically enabled in Xcode.

1.2.1 XIOS 2.0 (svn v1322)

Do the following:

```
mkdir XIOS
cd XIOS
svn checkout -r 1322 http://forge.ipsl.jussieu.fr/ioserver/svn/XIOS/trunk xios-2.0
```

Note: `svn checkout http://forge.ipsl.jussieu.fr/ioserver/svn/XIOS/branchs/xios-2.0 xios-2.0` also works with instructions below.

To get XIOS to compile, the compilers and packages need to be pointed to first, via modifying files in `arch`. Since I am using `gcc`, I did the following just to make a fresh copy:

```
cd xios2.0/arch
cp arch-GCC_LINUX.env arch-GCC_local.env
cp arch-GCC_LINUX.fcm arch-GCC_local.fcm
cp arch-GCC_LINUX.path arch-GCC_local.path
```

The `*.env` file specifies where HDF5 and NetCDF4 libraries live. The `*.fcm` file specifies which compilers and options to use. The `*.path` file specifies which paths and options to include. My files look like the following:

```
# arch-GCC_local.env

export HDF5_INC_DIR=/usr/local/include       # CHANGE ME
export HDF5_LIB_DIR=/usr/local/lib           # CHANGE ME

export NETCDF_INC_DIR=/usr/local/include     # CHANGE ME
export NETCDF_LIB_DIR=/usr/local/lib         # CHANGE ME
```

You could get an idea where the HDF5 and NetCDF4 directories are by doing `which h5copy` and `which nc-config` (assuming these are on `$PATH`), which should give you a `directory/bin`, and it is the `directory` part you want. If you did install the libraries somewhere else as in *other packages*, say, then make sure the `which` commands are pointing to the right place.

```
# arch-GCC_local.fcm

```

(continues on next page)

(continued from previous page)

```
################################################################################
##################                Projet XIOS               ##################
################################################################################

%CCOMPILER      /usr/local/bin/mpicc                # CHANGE ME
%FCOMPILER      /usr/local/bin/mpif90               # CHANGE ME
%LINKER         /usr/local/bin/mpif90               # CHANGE ME

%BASE_CFLAGS    -ansi -w
%PROD_CFLAGS    -O3 -DBOOST_DISABLE_ASSERTS
%DEV_CFLAGS     -g -O2
%DEBUG_CFLAGS   -g

%BASE_FFLAGS    -D__NONE__
%PROD_FFLAGS    -O3
%DEV_FFLAGS     -g -O2
%DEBUG_FFLAGS   -g

%BASE_INC       -D__NONE__
%BASE_LD        -lstdc++

%CPP            cpp-4.9                             # CHANGE ME
%FPP            cpp-4.9 -P                          # CHANGE ME
%MAKE           make
```

Check the MPI locations and versions by doing `which mpicc` and `mpicc --version` say. If they are the right ones you could just have `mpicc` instead of the full path as given above. MPI bindings are used here to avoid a possible error that may pop up in relation to the build trying to find `mpi.h`. The `gmake` command was swapped out by the `make` command (I don't have `cmake` on the laptop).

Note: For `gcc5.4` and maybe newer versions, doing just the above when compiling leads to a whole load of errors about clashing in C++:

```
.../include/boost/functional/hash/extensions.hpp:69:33: error: 'template<class T, class␣
↪A> std::size_t boost::hash_value' conflicts with a previous declaration
 std::size_t hash_value(std::list<T, A> const& v)
                                 ^
```

Adding `-D_GLIBCXX_USE_CXX11_ABI=0` to `%BASE_CFLAGS` fixes these.

```
# arch-GCC_local.path

NETCDF_INCDIR="-I$NETCDF_INC_DIR"
NETCDF_LIBDIR="-Wl,'--allow-multiple-definition' -L$NETCDF_LIB_DIR"
NETCDF_LIB="-lnetcdff -lnetcdf"

MPI_INCDIR=""
MPI_LIBDIR=""
MPI_LIB=""

HDF5_INCDIR="-I$HDF5_INC_DIR"
```

(continues on next page)

(continued from previous page)

```
HDF5_LIBDIR="-L$HDF5_LIB_DIR"
HDF5_LIB="-lhdf5_hl -lhdf5 -lhdf5 -lz"
```

The above has all the OASIS (the atmosphere / ocean coupler) keys removed. I added the `-Wl, '--allow-multiple-definition'` key for reasons I don't remember anymore...

I went into `bld.cfg`, found the line

```
bld::tool::cflags    %CFLAGS %CBASE_INC -I${PWD}/extern/src_netcdf -I${PWD}/
↪extern/boost/include -I${PWD}/extern/rapidxml/include -I${PWD}/extern/blitz/
↪include
```

and changed `src_netcdf` to `src_netcdf4` (see *XIOS1.0 stuff* for the reason).

Now it should be ready to compile. Assuming the current directory is `xios2.0/arch`:

```
cd ../
./make_xios --full --prod --arch GCC_local -j2 |& tee compile_log.txt
```

The `-j2` option uses two processors to build. The `tee` command is to keep logs of potential errors (the `|&` is short for `2>&1 |`) for debugging the compiler issues that may arise. It should work and takes around 5 mins to compile for me. The main end result is are binaries in `xios2.0/bin/` which NEMO will call.

1.2.2 NEMO 3.7/4.0 (svn v8666)

Check out a version of NEMO. I have another folder separate to the XIOS folders to contain the NEMO codes and binaries:

```
mkdir NEMO
cd NEMO
svn checkout -r 8666 http://forge.ipsl.jussieu.fr/nemo/svn/NEMO/trunk nemo3.7-8666
```

This checks out version 8666 (NEMO 3.7/4.0) and dumps it into a folder called `nemo3.7-8666` (change the target path to whatever you like). A similar procedure to specify compilers and where XIOS lives needs to be done for NEMO. Again, because of the compilers I am using:

```
cd nemo3.7-8666/NEMOGCM/ARCH
cp OLD/arch-gfortran_linux.fcm ./arch-gfortran_local.fcm
```

None of the fcm files associated with gfortran actually worked for me out of the box so here is my build of it (click HERE for a detailed log of how I got to the following):

```
# gfortran_local.fcm

# generic gfortran compiler options for linux
# NCDF_INC    netcdf include file
# NCDF_LIB    netcdf library
# FC          Fortran compiler command
# FCFLAGS     Fortran compiler flags
# FFLAGS      Fortran 77 compiler flags
# LD          linker
# LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
# FPPFLAGS    pre-processing flags
```

(continues on next page)

(continued from previous page)

```
# AR          assembler
# ARFLAGS     assembler flags
# MK          make
# USER_INC    additional include files for the compiler,  e.g. -I<include dir>
# USER_LIB    additional libraries to pass to the linker, e.g. -l<library>

%NCDF_HOME           /usr/local                                          # CHANGE ME

%XIOS_HOME           /home/julian/testing/gcc4.9-builds/XIOS/xios-2.0  # CHANGE ME

%CPP                 cpp-4.9                                             # CHANGE ME
%CPPFLAGS            -P -traditional

%XIOS_INC            -I%XIOS_HOME/inc
%XIOS_LIB            -L%XIOS_HOME/lib -lxios

%NCDF_INC            -I%NCDF_HOME/include
%NCDF_LIB            -L%NCDF_HOME/lib -lnetcdf -lnetcdff -lstdc++
%FC                  mpif90                                              # CHANGE ME
%FCFLAGS             -fdefault-real-8 -O3 -funroll-all-loops -fcray-pointer -cpp -ffree-
↪line-length-none
%FFLAGS              %FCFLAGS
%LD                  %FC
%LDFLAGS
%FPPFLAGS            -P -C -traditional
%AR                  ar
%ARFLAGS             -rs
%MK                  make
%USER_INC            %XIOS_INC %NCDF_INC
%USER_LIB            %XIOS_LIB %NCDF_LIB
```

The main changes are (see here for an attempt at the reasoning and a log of errors that motivates the changes):

- added `%NCDF_HOME` to point to where NetCDF lives
- added `%XIOS_*` keys to point to where XIOS lives
- added `%CPP` and flags, consistent with using `gcc4.9`
- added the `-lnetcdff` and `-lstdc++` flags to NetCDF flags
- using `mpif90` which is a MPI binding of `gfortran-4.9`
- added `-cpp` and `-ffree-line-length-none` to Fortran flags
- swapped out `gmake` with `make`

Then, I did (see *NEMO 3.6* for the reason):

```
cd ../CONFIG/
./makenemo -j0 -r GYRE_PISCES -n GYRE_testing -m gfortran_local
```

Edit `/GYRE_testing/cpp_GYRE_testing.fcm` and replaced `key_top` with `key_nosignedzero` (does not compile TOP for speed reasons, and make sure zeros are not signed). Then

```
./makenemo -j2 -n GYRE_testing -m gfortran_local |& tee compile_log.txt
```

This uses two processors, with GYRE as a reference, builds a new folder called GYRE_testing, with the specified architecture file, and outputs a log.

Note: The -r GYRE flag here only needs to be done once to create an extra folder and add GYRE_testing to cfg.txt. The subsequent compilations should then read, e.g., ./makenemo -n GYRE_testing -m gfortran_local.

Check that it does run with the following:

```
cd GYRE_testing/EXP00
mpiexec -n 1 ./opa
```

This may be mpirun instead of mpiexec, and -n 1 just runs it as a single core process. Change nn_itend = 4320 in nn_itend = 120 to only run it for 10 days (rdt = 7200 which is 2 hours). With all the defaults as is, there should be some GYRE_5d_*.nc data in the folder. You can read this with ncview (see the ncview page or, if you have sudo access, you can install it through sudo apt-get install ncview), bearing in mind that this is actually a rotated gyre configuration (see the following NEMO forge page or search for gyre in the NEMO book).

Note: If your installation compiles but does not run with the following error

```
dyld: Library not loaded: @rpath/libnetcdff.6.dylib
Referenced from: /paths/./nemo
Reason: no suitable image found.  Did find:
/usr/local/lib/libnetcdff.6.dylib: stat() failed with errno=13
```

then it is not finding the right libraries. These could be fixed by adding the -Wl,-rpath,/fill me in/lib flag to the relevant flags bit in the *.fcm files (or possibly in XIOS the path and/or env) to specify exactly where the libraries live. This can happen for example on a Mac or if the libraries are installed not at the usual place.

Note: One infuriating problem I had specifically with a Mac (though it might be a gcc4.8 issue) is that the run does not get beyond the initialisation stage. Going into ocean.output and searching for E R R O R shows that it complained about a misspelled namelist item (in my case it was in the namberg namelist). If you go into output.namelist.dyn and look for the offending namelist is that it might be reading in nonsense. This may happen if the comment character ! is right next to a variable, e.g.

```
ln_icebergs = .true.!this is a comment
```

Fix this by adding a white space, i.e.

```
ln_icebergs = .true. !this is a comment
```

1.3 NEMO 4.0 (beta) + XIOS 2.5

Tested with

- `gcc4.9`, `gcc5.4` on a linux system
- `gcc4.8` on a Mac (El Capitan OSX 10.11)

The code structure in NEMO 4.0 and the use of some commands are slightly different (at least in v9925) and will be documented below (please see the official NEMO annoucement for details). If you get errors that are not documented here, see if *the XIOS1.0 NEMO3.6* page contains the relevant errors.

The assumption here is that the compiler is fixed and the packages (e.g., NetCDF4 and a MPI bindings) are configured to be consistent with the compilers. See *here* to check whether the binaries exist, where they are, and how they might be installed separately if need be. All the `#CHANGE ME` highlighted below needs be modified to point to the appropriate paths or binaries (soft links with `ln -s` are ok).

The instructions below uses `gcc4.9` for demonstration (modifications with `gcc5.4` as appropriate). I defined some extra variables on a Linux machine:

```
export $BD=/home/julian/testing/gcc4.9-builds # CHANGE ME

export C_INCLUDE_PATH=$BD/install/include:$C_INCLUDE_PATH
export CPLUS_INCLUDE_PATH=$BD/install/include:$CPLUS_INCLUDE_PATH
export LIBRARY_PATH=$BD/install/lib:$LIBRARY_PATH
export LD_LIBRARY_PATH=$BD/install/lib:$LD_LIBRARY_PATH
```

You shouldn't need to do the above if the packages are forced to look at the right place (e.g. via `-L` and/or `-I` flags with path to libraries and include files respectively). Not all of these are necessary depending on whether you choose to build/have static or dynamic libraries, and the `LD_LIBRARY_PATH` seems to sort out a lot of problems with linking libraries.

On a Mac done through anaconda the above was not necessary. My understanding is that setting these variables might not actually do anything unless an option is specifically enabled in Xcode.

1.3.1 XIOS 2.5 (svn v1566)

Note: Looks like you could use XIOS 2.0 with NEMO 4.0, so if the following doesn't work for you, try compiling *XIOS 2.0* instead.

Do the following:

```
mkdir XIOS
cd XIOS
svn checkout -r 1566 http://forge.ipsl.jussieu.fr/ioserver/svn/XIOS/branchs/xios-2.5␣
↪xios-2.5
```

To get XIOS to compile, the compilers and packages need to be pointed to first, via modifying files in `arch`. Since I am using `gcc`, I did the following just to make a fresh copy:

```
cd xios2.5/arch
cp arch-GCC_LINUX.env arch-GCC_local.env
cp arch-GCC_LINUX.fcm arch-GCC_local.fcm
cp arch-GCC_LINUX.path arch-GCC_local.path
```

The `*.env` file specifies where HDF5 and NetCDF4 libraries live. The `*.fcm` file specifies which compilers and options to use. The `*.path` file specifies which paths and options to include. My files look like the following:

```
# arch-GCC_local.env

export HDF5_INC_DIR=/usr/local/include       # CHANGE ME
export HDF5_LIB_DIR=/usr/local/lib           # CHANGE ME

export NETCDF_INC_DIR=/usr/local/include     # CHANGE ME
export NETCDF_LIB_DIR=/usr/local/lib         # CHANGE ME
```

You could get an idea where the HDF5 and NetCDF4 directories are by doing `which h5copy` and `which nc-config` (assuming these are on `$PATH`), which should give you a `directory/bin`, and it is the `directory` part you want. If you did install the libraries somewhere else as in *other packages*, say, then make sure the `which` commands are pointing to the right place.

```
# arch-GCC_local.fcm

############################################################################
###################                Projet XIOS               ###################
############################################################################

%CCOMPILER      /usr/local/bin/mpicc                # CHANGE ME
%FCOMPILER      /usr/local/bin/mpif90               # CHANGE ME
%LINKER         /usr/local/bin/mpif90               # CHANGE ME

%BASE_CFLAGS    -ansi -w
%PROD_CFLAGS    -O3 -DBOOST_DISABLE_ASSERTS
%DEV_CFLAGS     -g -O2
%DEBUG_CFLAGS   -g

%BASE_FFLAGS    -D__NONE__
%PROD_FFLAGS    -O3
%DEV_FFLAGS     -g -O2
%DEBUG_FFLAGS   -g

%BASE_INC       -D__NONE__
%BASE_LD        -lstdc++

%CPP            cpp-4.9                             # CHANGE ME
%FPP            cpp-4.9 -P                          # CHANGE ME
%MAKE           make
```

Check the MPI locations and versions by doing `which mpicc` and `mpicc --version` say. If they are the right ones you could just have `mpicc` instead of the full path as given above. MPI bindings are used here to avoid a possible error that may pop up in relation to the build trying to find `mpi.h`. The `gmake` command was swapped out by the `make` command (I don't have `cmake` on the laptop).

Note: For `gcc5.4` and maybe newer versions, doing just the above when compiling leads to a whole load of errors about clashing in C++:

```
.../include/boost/functional/hash/extensions.hpp:69:33: error: 'template<class T, class␣
↪A> std::size_t boost::hash_value' conflicts with a previous declaration
```

(continues on next page)

(continued from previous page)

```
 std::size_t hash_value(std::list<T, A> const& v)
                                ^
```

Adding `-D_GLIBCXX_USE_CXX11_ABI=0` to `%BASE_CFLAGS` fixes these.

A difference I've found between XIOS 2.5 and other XIOS versions is that doing just the above might lead to an error like the following:

```
This file requires compiler and library support for the ISO C++ 2011 standard. This␣
↪support is currently experimental, and must be enabled with the -std=c++11 or -
↪std=gnu++11 compiler options.
```

Adding `-std=c++11` to `%BASE_CFLAGS` seems to fix this.

You might also get the following:

```
SUBROUTINE cxios_set_interpolate_domain_read_write_convention(interpolate_domain_hdl,␣
↪read_write_convention, read_write_conventi
                                                                                ␣
↪                                          1
Error: Unexpected junk in formal argument list at (1)
```

The Fortran lines are too long, so fix this by adding `-ffree-line-length-none` to `%BASE_FFLAGS`.

```
# arch-GCC_local.path

NETCDF_INCDIR="-I$NETCDF_INC_DIR"
NETCDF_LIBDIR="-Wl,'--allow-multiple-definition' -L$NETCDF_LIB_DIR"
NETCDF_LIB="-lnetcdff -lnetcdf"

MPI_INCDIR=""
MPI_LIBDIR=""
MPI_LIB=""

HDF5_INCDIR="-I$HDF5_INC_DIR"
HDF5_LIBDIR="-L$HDF5_LIB_DIR"
HDF5_LIB="-lhdf5_hl -lhdf5 -lhdf5 -lz"
```

The above has all the OASIS (the atmosphere / ocean coupler) keys removed. I added the `-Wl, '--allow-multiple-definition'` key for reasons I don't remember anymore…

I went into `bld.cfg`, found the line

```
bld::tool::cflags    %CFLAGS %CBASE_INC -I${PWD}/extern/src_netcdf -I${PWD}/
↪extern/boost/include -I${PWD}/extern/rapidxml/include -I${PWD}/extern/blitz/
↪include
```

and changed `src_netcdf` to `src_netcdf4` (see *XIOS1.0 stuff* for the reason).

Now it should be ready to compile. Assuming the current directory is `xios2.5/arch`:

```
cd ../
./make_xios --full --prod --arch GCC_local -j2 |& tee compile_log.txt
```

The `-j2` option uses two processors to build. The `tee` command is to keep logs of potential errors (the `|&` is short for `2>&1 |`) for debugging errors that may arise.

1.3.2 NEMO 4.0 (svn v9925)

There is a restructuring of folders (see the official annoucement for details) so the commands below will reflect this.

Check out a version of NEMO. I have another folder separate to the XIOS folders to contain the NEMO codes and binaries:

```
mkdir NEMO
cd NEMO
svn checkout -r 9925 http://forge.ipsl.jussieu.fr/nemo/svn/NEMO/trunk nemo4.0-9925
```

This checks out version 9925 (NEMO 4.0 beta) and dumps it into a folder called `nemo4.0-9925` (change the target path to whatever you like).

Note: `svn checkout https://forge.ipsl.jussieu.fr/nemo/svn/NEMO/releases/release-4.0 nemo4.0` would pull the official version

A similar procedure to specify compilers and where XIOS lives needs to be done for NEMO. Again, because of the compilers I am using:

```
cd nemo4.0-9925/arch
cp arch-linux_gfortran.fcm ./gfortran_local.fcm
```

None of the fcm files associated with gfortran actually worked for me out of the box so here is my build of it (click HERE for a detailed log of how I got to the following):

```
# gfortran_local.fcm

# generic gfortran compiler options for linux
# NCDF_INC    netcdf include file
# NCDF_LIB    netcdf library
# FC          Fortran compiler command
# FCFLAGS     Fortran compiler flags
# FFLAGS      Fortran 77 compiler flags
# LD          linker
# LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
# FPPFLAGS    pre-processing flags
# AR          assembler
# ARFLAGS     assembler flags
# MK          make
# USER_INC    additional include files for the compiler,  e.g. -I<include dir>
# USER_LIB    additional libraries to pass to the linker, e.g. -l<library>

%NCDF_HOME           /usr/local                                          # CHANGE ME

%XIOS_HOME           /home/julian/testing/gcc4.9-builds/XIOS/xios-2.5  # CHANGE ME

%CPP                 cpp-4.9                                             # CHANGE ME
%CPPFLAGS            -P -traditional
```

(continues on next page)

(continued from previous page)

```
%XIOS_INC            -I%XIOS_HOME/inc
%XIOS_LIB            -L%XIOS_HOME/lib -lxios

%NCDF_INC            -I%NCDF_HOME/include
%NCDF_LIB            -L%NCDF_HOME/lib -lnetcdf -lnetcdff -lstdc++
%FC                  mpif90                                          # CHANGE ME
%FCFLAGS             -fdefault-real-8 -O3 -funroll-all-loops -fcray-pointer -cpp -ffree-
↪line-length-none
%FFLAGS              %FCFLAGS
%LD                  %FC
%LDFLAGS
%FPPFLAGS            -P -C -traditional
%AR                  ar
%ARFLAGS             -rs
%MK                  make
%USER_INC            %XIOS_INC %NCDF_INC
%USER_LIB            %XIOS_LIB %NCDF_LIB
```

The main changes are (see here for an attempt at the reasoning and a log of errors that motivates the changes):

- added `%NCDF_HOME` to point to where NetCDF lives
- added `%XIOS_*` keys to point to where XIOS lives
- added `%CPP` and flags, consistent with using `gcc4.9`
- added the `-lnetcdff` and `-lstdc++` flags to NetCDF flags
- using `mpif90` which is a MPI binding of `gfortran-4.9`
- added `-cpp` and `-ffree-line-length-none` to Fortran flags
- swapped out `gmake` with `make`

Go into the configuration folder by

```
cd ../cfgs
```

One of the things I noticed is that `makenemo` now seems to work slightly differently (at least with this version). Normally you can do `makenemo -r GYRE -n GYRE_testing -j0 -m gcc_fortran_local`, which copies a configuration but does not compile it, so you can edit the `cpp` flags before compiling (and note that it adds an entry into `works_cfgs.txt`). However now it seems you have to specify a `-r` flag or a `-d` flag (which specifies what NEMO modules the configuration should have), whereas before just a `-n` flag would work by itself.

You could just compile as usual with `makenemo` (see *NEMO 3.6* for syntax). The slightly untidy way to circumvent errors that I know will come up was to do the following:

1. Open `refs_cfg.txt`, copy the `GYRE_PISCES OCE TOP` line and paste it at the bottom, but then change the configuration name (`GYRE_PISCES` to `GYRE_testing` in my case), save and close it;
2. Then do

   ```
   mkdir GYRE_testing
   rsync -arv GYRE_PISCES/* GYRE_testing/
   ```

3. I opened `/GYRE_testing/cpp_GYRE_testing.fcm` and replaced `key_top` with `key_nosignedzero` (does not compile TOP for speed speeds, and make sure zeros are not signed), save it;

4. Compile with (because `makenmemo` is now one level up)

```
../makenemo -j2 -r GYRE_testing -m gfortran_local |& tee compile_log.txt
```

(note the `-r` rather than `-n` flag here).

Warning: See if this feature of `makenemo` has been modified in the trunk?

Note the executable `opa` is now called `nemo` (so make sure you change those submission scripts on the relevant clusters if you use NEMO on them). Check that it does run with the following:

```
cd GYRE_testing/EXP00
mpiexec -n 1 ./nemo
```

Note that what used to be `solver.stat` is now called `run.stat`, and there is an extra `run.stat.nc` for whatever reason. The `ocean.output` file is still the same.

Note: If your installation compiles but does not run with the following error

```
dyld: Library not loaded: @rpath/libnetcdff.6.dylib
Referenced from: /paths/./nemo
Reason: no suitable image found.  Did find:
/usr/local/lib/libnetcdff.6.dylib: stat() failed with errno=13
```

then it is not finding the right libraries. These could be fixed by adding the `-Wl,-rpath,/fill me in/lib` flag to the relevant flags bit in the `*.fcm` files (or possibly in XIOS the `path` and/or `env`) to specify exactly where the libraries live. This can happen for example on a Mac or if the libraries are installed not at the usual place.

Note: One infuriating problem I had specifically with a Mac (though it might be a `gcc4.8` issue) is that the run does not get beyond the initialisation stage. Going into `ocean.output` and searching for `E R R O R` shows that it complained about a misspelled namelist item (in my case it was in the `namberg` namelist). If you go into `output.namelist.dyn` and look for the offending namelist is that it might be reading in nonsense. This may happen if the comment character `!` is right next to a variable, e.g.

```
ln_icebergs = .true.!this is a comment
```

Fix this by adding a white space, i.e.

```
ln_icebergs = .true. !this is a comment
```

1.4 NEMO 4.2 + XIOS 2.5

Tested with

- `gcc8.3.0` on a computer cluster (HPC3, with in-built parallel HDF5 and NetCDF4)

The new official page is here and here. Following the instruction there largely works; below details minor things I needed to fix on the particular machine I tested on.

1.4.1 XIOS 2.5 (svn v2462)

According to the NEMO install guide we should use the trunk of XIOS, so

```
svn co http://forge.ipsl.jussieu.fr/ioserver/svn/XIOS/trunk
```

(the version I happen to get is `v2462`). Previously I had issues with newer GCC versions that seems to have been circumvented somehow, and XIOS builds with adding `-D_GLIBCXX_USE_CXX11_ABI=0` and `-std=c++11` to the `BASE_CFLAGS`.

```
# arch-HKUST_HPC3.fcm

################################################################################
###################                Projet XIOS               ###################
################################################################################

%CCOMPILER      mpicc                 # CHANGE ME
%FCOMPILER      mpif90                # CHANGE ME
%LINKER         mpif90                # CHANGE ME

%BASE_CFLAGS    -ansi -w -D_GLIBCXX_USE_CXX11_ABI=0 -std=c++11
%PROD_CFLAGS    -O3 -DBOOST_DISABLE_ASSERTS
%DEV_CFLAGS     -g -O2
%DEBUG_CFLAGS   -g

%BASE_FFLAGS    -D__NONE__ -ffree-line-length-none
%PROD_FFLAGS    -O3
%DEV_FFLAGS     -g -O2
%DEBUG_FFLAGS   -g

%BASE_INC       -D__NONE__
%BASE_LD        -lstdc++

%CPP            cpp                               # CHANGE ME
%FPP            cpp -P                            # CHANGE ME
%MAKE           make
```

On HPC3 the various NetCDF4 folders are build and dumped in separate folders, so something needed to be done to the `env` and `path` variable entries, as follows (making sure to load the modules accordingly):

```
# arch-HKUST_HPC3.path

NETCDF_INCDIR="-I $NETCDF_INC_DIR -I $NETCDFF_INC_DIR"
NETCDF_LIBDIR="-L $NETCDF_LIB_DIR -L $NETCDFF_LIB_DIR"
```

(continues on next page)

(continued from previous page)

```
NETCDF_LIB="-lnetcdff -lnetcdf"

HDF5_INCDIR="-I $HDF5_INC_DIR"
HDF5_LIBDIR="-L $HDF5_LIB_DIR"
HDF5_LIB="-lhdf5_hl -lhdf5 -lhdf5 -lz"
```

```
# arch-HKUST_HPC3.env

export HDF5_INC_DIR=/opt/ohpc/pub/libs/gnu8/openmpi3/hdf5/1.10.5/include
export HDF5_LIB_DIR=/opt/ohpc/pub/libs/gnu8/openmpi3/hdf5/1.10.5/lib

export NETCDF_INC_DIR=/opt/ohpc/pub/libs/gnu8/openmpi3/netcdf/4.7.1/include
export NETCDF_LIB_DIR=/opt/ohpc/pub/libs/gnu8/openmpi3/netcdf/4.7.1/lib

export NETCDFF_INC_DIR=/opt/ohpc/pub/libs/gnu8/openmpi3/netcdf-fortran/4.5.2/include
export NETCDFF_LIB_DIR=/opt/ohpc/pub/libs/gnu8/openmpi3/netcdf-fortran/4.5.2/lib
```

I went into `bld.cfg`, found the line

```
bld::tool::cflags    %CFLAGS %CBASE_INC -I${PWD}/extern/src_netcdf -I${PWD}/
↪extern/boost/include -I${PWD}/extern/rapidxml/include -I${PWD}/extern/blitz/
↪include
```

and changed `src_netcdf` to `src_netcdf4` (see *XIOS1.0 stuff* for the reason). Then compile as usual:

```
cd ../
./make_xios --full --prod --arch GCC_local -j2 |& tee compile_log.txt
```

1.4.2 NEMO 4.2 (Git SHA 216c746957a674552de5bf02c17d22fa37f2a0d4)

NEMO is as of writing no longer using SVN, and managing code through Git instead. So I downloaded it by

```
git clone https://forge.nemo-ocean.eu/nemo/nemo.git nemo_4.2.0
```

I downloaded the whole thing and then looked to switch branches. To get only the official release, add the flag `-b 4.2.0` (or download the whole thing and then switch using `git switch --detach 4.2.0`). After some trial and error I basically did

```
# arch-HKUST_HPC3.fcm

%XIOS_HOME           /scratch/PI/jclmak/XIOS_mpi/xios-2.5-r2462

%CPP                 cpp
%CPPFLAGS            -P -traditional

%XIOS_INC            -I%XIOS_HOME/inc
%XIOS_LIB            -L%XIOS_HOME/lib -lxios

%NCDF_INC            -I/opt/ohpc/pub/libs/gnu8/openmpi3/netcdf-fortran/4.5.2/include -I/
↪opt/ohpc/pub/libs/gnu8/openmpi3/netcdf/4.7.1/include
%NCDF_LIB            -L/opt/ohpc/pub/libs/gnu8/openmpi3/netcdf/4.7.1/lib -L/opt/ohpc/pub/
```

(continues on next page)

(continued from previous page)

```
↪libs/gnu8/openmpi3/netcdf-fortran/4.5.2/lib -lnetcdf -lnetcdff -lstdc++
%FC                   mpif90
%FCFLAGS              -fdefault-real-8 -O3 -funroll-all-loops -fcray-pointer -cpp -ffree-
↪line-length-none
%FFLAGS               %FCFLAGS
%LD                   %FC
%LDFLAGS
%FPPFLAGS             -P -C -traditional
%AR                   ar
%ARFLAGS              -rs
%MK                   make
%USER_INC             %XIOS_INC %NCDF_INC
%USER_LIB             %XIOS_LIB %NCDF_LIB
```

and everything built fine. The tricky bit was the combination of module loads, and I went one step further and brute force pointed to the relevant `include` and `lib` folders.

Note: I had some issues with using older compilers and/or OpenMPI. XIOS will compile fine, but when compiling NEMO experiments will lead to something like

```
There is no specific subroutine for the generic 'mpi_dist_graph_create_adjacent'
```

Hence the new test compile with newer compilers (because this was the one that already interfaces with the newer OpenMPI3).

The usage is as in *NEMO 4.0*.

Note: The zenodo repository when I went to check (`ORCA2_ICE_v4.2.tar`) for the inputs when testing `ORCA2` was missing stuff (e.g. `iwd`, internal wave dissipation probably), so I just went into `namelist_cfg` and switched it off, and it run as usual.

1.5 Oxford ARC compilation

The build uses NEMO 3.7/4.0 + XIOS 2.0 as the example. For installing other versions, extrapolate from the other notes.

Annoyingly (!) everything basically works out of the box because all the dependency modules have been built already! This has not been the usual experience I have with XIOS and NEMO…

1.5.1 Building NEMO and XIOS

Log on first using:

```
ssh [-X] phys????@arcus-b.arc.ox.ac.uk
```

On first login doing `module list` should show no modules; if there are then might want to do `module purge` just for safety. Doing `module avail` shows the list of modules available. I'm going to use the `gcc` one, and by doing

```
module load /netcdf-parallel/4.4__mvapich2__gcc
```

this loads NetCDF4 as well as its dependencies (which should be HDF5, gcc4.9.2 and the relevant mvapich); I added that line to `~/.bashrc` so it loads from now on when logging in. Then I did

```
echo $LD_LIBRARY_PATH
> /system/software/arcus-b/lib/netcdf/4.4/mvapich2-2.1.0__gcc-4.9.2/lib:/system/software/
↪arcus-b/lib/hdf5/1.8.12/mvapich2-2.1.0__gcc-4.9.2/lib ...
```

which tells me where the NetCDF and HDF5 libraries live. So for XIOS I do

```
cd $DATA # <--- this is the "work" directory (which is generically not ~/)
mkdir XIOS
cd XIOS
svn co http://forge.ipsl.jussieu.fr/ioserver/svn/XIOS/trunk@1322 xios-2.0
cd xios2.0/arch
cp arch-GCC_LINUX.env arch-GCC_ARC.env
cp arch-GCC_LINUX.fcm arch-GCC_ARC.fcm
cp arch-GCC_LINUX.path arch-GCC_ARC.path
```

with

```
# arch-GCC_ARC.env

export HDF5_INC_DIR=/system/software/arcus-b/lib/hdf5/1.8.12/mvapich2-2.1.0__gcc-4.9.2/
↪include
export HDF5_LIB_DIR=/system/software/arcus-b/lib/hdf5/1.8.12/mvapich2-2.1.0__gcc-4.9.2/
↪lib

export NETCDF_INC_DIR=/system/software/arcus-b/lib/netcdf/4.4/mvapich2-2.1.0__gcc-4.9.2/
↪include
export NETCDF_LIB_DIR=/system/software/arcus-b/lib/netcdf/4.4/mvapich2-2.1.0__gcc-4.9.2/
↪lib
```

and the other two as default. Running

```
cd ../
./make_xios --full --prod --arch HKUST_HPC2 -j4 |& tee compile_log.txt
```

seems to do the job. I think I did go into `bld.cfg` and changed `src_netcdf` to `src_netcdf4` for safety; don't remember needing this in ARCHER (did need it when doing a local compilation). If that doesn't work consider adding `CPPFLAGS` and `LDFLAGS` before the `./make_xios` command to force the program to look in the specified place.

NEMO is then built as follows:

```
cd $DATA
mkdir NEMO
cd NEMO
svn co http://forge.ipsl.jussieu.fr/nemo/svn/NEMO/trunk@8666 nemo3.7-8666
cd nemo3.7-8666/NEMOGCM/ARCH
cp OLD/arch-gfortran_linux.fcm ./arch-GCC_ARC.fcm
```

using

```
# arch-GCC_ARC.fcm
# generic gfortran compiler options for linux
# NCDF_INC    netcdf include file
# NCDF_LIB    netcdf library
# FC          Fortran compiler command
# FCFLAGS     Fortran compiler flags
# FFLAGS      Fortran 77 compiler flags
# LD          linker
# LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
# FPPFLAGS    pre-processing flags
# AR          assembler
# ARFLAGS     assembler flags
# MK          make
# USER_INC    additional include files for the compiler,  e.g. -I<include dir>
# USER_LIB    additional libraries to pass to the linker, e.g. -l<library>

%XIOS_HOME           $DATA/XIOS/xios-2.0

%CPP                 cpp
%CPPFLAGS            -P -traditional

%XIOS_INC            -I%XIOS_HOME/inc
%XIOS_LIB            -L%XIOS_HOME/lib -lxios

%NCDF_INC            -I/system/software/arcus-b/lib/netcdf/4.4/mvapich2-2.1.0__gcc-4.9.2/
↪include
%NCDF_LIB            -L/system/software/arcus-b/lib/netcdf/4.4/mvapich2-2.1.0__gcc-4.9.2/
↪lib -lnetcdf -lnetcdff -lstdc++
%FC                  mpif90
%FCFLAGS             -fdefault-real-8 -O3 -funroll-all-loops -fcray-pointer -cpp -ffree-
↪line-length-none
%FFLAGS              %FCFLAGS
%LD                  %FC
%LDFLAGS
%FPPFLAGS            -P -C -traditional
%AR                  ar
%ARFLAGS             -rs
%MK                  make
%USER_INC            %XIOS_INC %NCDF_INC
%USER_LIB            %XIOS_LIB %NCDF_LIB
```

followed by

```
cd ../CONFIG
```

(continues on next page)

(continued from previous page)

```
./makenemo -r GYRE_PISCES -n GYRE_testing -m GCC_ARC -j0
nano GYRE_testing/cpp_GYRE_testing.fcm # (have key_top -> key_nosignedzero)
./makenemo -n GYRE_tesitng -m GCC_ARC -j4
```

and it should work. One more thing we will do is to make `TOOLS/REBUILD_NEMO`:

```
cd ../TOOLS
./maketools -n REBUILD_NEMO -m GCC_ARC
```

1.5.2 Running NEMO on the ARC

The system uses SLURM and the key commands are

- `sbatch [submit_nemo]`: submits the job detailed in `submit_nemo` (see below)
- `scancel [job ID]`: cancel the job
- `sinfo`: check status of queues available
- `squeue -u $USER`: check job info for `$USER`

`sbatch` could be used with arguments but I am going to have everything within `submit_nemo` itself. Check balance and budget account names with the `mybalance` command. Running `sinfo` shows the queue available is called `compute`. One thing to note is that ARC has 16 cores per node and this is reflected in the core/node request numbers.

Oxford ARC does have parallel NetCDF so I can use XIOS in detached mode. To do this I link `xios_server.exe` to the folder:

```
cd GYRE_testing/EXP00
ln -s $DATA/XIOS/xios2.0/bin/xios_server.exe .
```

Modify `iodef.xml` so that the user server boolean is `true`. Additionally I go into `file_def_nemo.xml` and swap out `multiple_file` at the top header to `one_file`, which then spits out a single NetCDF file. This however only works for the diagnostic files but not the restart files, so recombining the restart files we are going to call `TOOLS/REBUILD_NEMO` in the post-processing script.

The generic submission script I use (based on the one given on the NOCL page) is as follows (I have some ASCII art in there because I got bored at some point):

```
#!/bin/bash

# NOTE: Lines starting with "#SBATCH" are valid SLURM commands or statements,
#       while those starting with "#" and "##SBATCH" are comments.  Uncomment
#       "##SBATCH" line means to remove one # and start with #SBATCH to be a
#       SLURM command or statement.

#=================================================================
# DEFINE SOME JUNK FOR THE SUBMISSION (??? make this more flexible with e.g. queues?)
#=================================================================

#SBATCH -J gyre04       # job name
#SBATCH -o stdouterr    # output and error file name
#SBATCH -n 32           # total number of mpi tasks requested
#SBATCH -N 2            # total number of nodes requested
#SBATCH -p compute      # queue (partition) -- standard, development, etc.
```

(continues on next page)

(continued from previous page)

```
#SBATCH -t 12:00:00     # maximum runtime

# Enable email notificaitons when job begins and ends, uncomment if you need it
##SBATCH --mail-user=user_name@ust.hk #Update your email address
##SBATCH --mail-type=begin
##SBATCH --mail-type=end

# Setup runtime environment if necessary
module purge
module load netcdf-parallel/4.4__mvapich2__gcc

#============================================================
# LAUNCH JOB
#============================================================

echo " _ __   ___ _ __ ___   ___          "
echo "| '_ \ / _ \ '_ ' _ \ / _ \         "
echo "| | | |  __/ | | | | | (_) |        "
echo "|_| |_|\___|_| |_| |_|\___/  v3.7   "

# Go to the job submission directory and run your application
cd /data/phys-geometric/phys1342/NEMO/nemo3.7-8666/NEMOGCM/CONFIG/GYRE_testing/EXP00
mpirun -n 2 ./xios_server.exe : -n 30 ./opa

#============================================================
# POSTPROCESSING
#============================================================

# kills the daisy chain if there are errors

if grep -q 'E R R O R' ocean.output ; then

  echo "E R R O R found, exiting..."
  echo "  ___ _ __ _ __ ___  _ __   "
  echo " / _ \ '__| '__/ _ \| '__|  "
  echo "|  __/ |  | | | (_) | |     "
  echo " \___|_|  |_|  \___/|_|     "
  echo "check out ocean.output or stdouterr to see what the deal is "

  exit
else
  echo "going into postprocessing stage..."
  # cleans up files, makes restarts, moves files, resubmits this pbs

  bash ./postprocess.sh >& cleanup.log
  exit
fi
```

The ratio of `XIOScore` to `NEMOcore` I never found to lead to major differences for the size of runs I do (not larger than 300 cores); vaguely remember reading somewhere that `XIOScore` hovering between 5 to 10 per cent of `NEMOcore` is ok.

The following post-processing script requires a few prepping (I make no apologies for the bad code and the script being

fickle; feel free to modify as you see fit):

- copying the `nn_date0` line into `namelist_cfg` from say `namelist_ref` if it doesn't exist already, because the time-stamps are modified by modifying `nn_date0`
- do a search in `namelist_cfg` and make sure there is only ever one mention of `nn_date0` (otherwise it grabs the wrong lines)
- `nn_date0` should not begin with zeros (e.g. `10101` rather than `010101` in `yymmdd`)
- in the experiment folder, do `mkdir RESTARTS OUTPUTS` (otherwise there is no folder to copy into)

The `postprocess.sh` I cooked up is here:

```
#!/bin/bash
#! postprocess.sh
#! Script to clean up the NEMO outputs

export BASE_DIR=$DATA/NEMO/nemo3.7-8666/NEMOGCM/
export MODEL=GYRE
export NUM_CPU=30

# time-stamp increment, yymmdd
export DATE_INC=100000

# when to stop the daisy chaining, yymmdd
export THRESH=10

# error catching (only when restart files etc cannot be copied or made)
export ERR_CATCH=0

#############################################################
# 0) recombine files to one netcdf (restarts and/or outputs)
# restarts: extract the restart file time-step stamp
#              based on the *0000.nc restart which should (!) always exist
#           rebuild the restart file in the submission directory
# outputs:  put them in manually and just do a grab
#           this assumes only files at the current time-stamp is there,
#              otherwise it will bug out as it grabs wrong files
#############################################################

# restart files
export RES_TIMESTAMP=$(echo $(ls -d ${MODEL}_*_restart_0000.nc) | awk -F _ '{print $2 }')

$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${RES_TIMESTAMP}_restart $NUM_CPU
if (($? > 0)); then
  ERR_CATCH=$((ERR_CATCH + 1))
  echo "  ERR: making the restart file in the folder"
fi
##$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${RES_TIMESTAMP}_restart_ice $NUM_
↪CPU

# output files (assumes a grid_T always exists)
#export OUT_FREQ=$(echo $(ls -d ${MODEL}_*_grid_T_0000.nc) | awk -F _ '{print $2 }')
#export OUT_START=$(echo $(ls -d ${MODEL}_*_grid_T_0000.nc) | awk -F _ '{print $3 }')
#export OUT_END=$(echo $(ls -d ${MODEL}_*_grid_T_0000.nc) | awk -F _ '{print $4 }')
```

(continues on next page)

(continued from previous page)

```
#$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${OUT_FREQ}_${OUT_START}_${OUT_END}_
↪grid_T $NUM_CPU
#$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${OUT_FREQ}_${OUT_START}_${OUT_END}_
↪grid_U $NUM_CPU
#$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${OUT_FREQ}_${OUT_START}_${OUT_END}_
↪grid_V $NUM_CPU
#$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${OUT_FREQ}_${OUT_START}_${OUT_END}_
↪grid_W $NUM_CPU

# add more things in here if output freqs are different etc

#########################################################
# 1) pull out some variables to modify namelist file
#########################################################

# pull the number out
# add the increment to it for new date
# subtract appropriately to get the date stamp
#   (e.g. 110101 - 8871 = 101230) and bulk out zeros

export OLD_DATE_STR=$(grep -ri "nn_date0" namelist_cfg)
export OLD_DATE_NUM=$(echo ${OLD_DATE_STR} | sed -e 's/[^0-9 ]//g' | awk '{print $NF}')
export NEW_DATE_NUM=$((OLD_DATE_NUM + DATE_INC))

# 8871 for 30 days a month (so the RES_STAMP=yyyy1230)
# otherwise do 8870        (so the RES_STAMP=yyyy1231)
# do something else for other time units
export RES_STAMP=$(printf %08d $((NEW_DATE_NUM - 8871)))

#########################################################
# 2) move files around and tidy up
#########################################################

cp -pv ${MODEL}_${RES_TIMESTAMP}_restart.nc ./RESTARTS/${MODEL}_${RES_STAMP}_restart.nc
cp -pv ./output.namelist.dyn ./OUTPUTS/output.namelist.dyn.${RES_STAMP}
#cp -pv ${MODEL}_${RES_TIMESTAMP}_restart_ice.nc ./RESTARTS/${MODEL}_${RES_STAMP}_
↪restart_ice.nc
#cp -pv ./output.namelist.ice ./OUTPUTS/output.namelist.ice.${RES_STAMP}
cp -pv ./ocean.output ./OUTPUTS/ocean.output.${RES_STAMP}
cp -pv ./solver.stat ./OUTPUTS/solver.stat.${RES_STAMP}
cp -pv ./stdouterr ./OUTPUTS/stdouterr.${RES_STAMP}
cp -pv ./namelist_cfg ./OUTPUTS/namelist_cfg.${RES_STAMP}

#cp -pv ./volume_transport ./OUTPUTS/volume_transport.${RES_STAMP}
#cp -pv ./salt_transport ./OUTPUTS/salt_transport.${RES_STAMP}
#cp -pv ./heat_transport ./OUTPUTS/heat_transport.${RES_STAMP}

rm -v ${MODEL}_${RES_TIMESTAMP}_restart*
rm -v restart.nc
#rm -v restart_ice.nc
rm -v ${MODEL}_*_????.nc
```

(continues on next page)

(continued from previous page)

```
mv ${MODEL}*.nc ./OUTPUTS

cp -pv RESTARTS/${MODEL}_${RES_STAMP}_restart.nc ./restart.nc
if (($? > 0)); then
  ERR_CATCH=$((ERR_CATCH + 1))
  echo "  ERR: copying restart file into folder"
fi

#cp -pv RESTARTS/${MODEL}_${RES_STAMP}_restart_ice.nc ./restart_ice.nc
#if (($? > 0)); then
#  ERR_CATCH=$((ERR_CATCH + 1))
#  echo "  ERR: copying restart_ice file into folder"
#fi

##########################################################
# 3) if all good, then modify namelist_cfg and resbumit
##########################################################

if (($ERR_CATCH > 0)) || ((${NEW_DATE_NUM} > $THRESH)); then
  if (($ERR_CATCH > 0)); then
    echo " "
    echo " "
    echo " "
    echo "ERR: caught a non-zero exit status, check cleanup.log for what the deal was"
    echo "ERR: caught a non-zero exit status, check cleanup.log for what the deal was"
  else
    echo "OK: grabbed time stamp ${NEW_DATE_NUM} larger than threshold ${THRESH},␣
↪breaking..."
    echo "OK: grabbed time stamp ${NEW_DATE_NUM} larger than threshold ${THRESH},␣
↪breaking..."
    # WARNING: this assumes that OLD_DATE_NUM is the only number within the file, which␣
↪should
    #          really be true
    sed -i "s/${OLD_DATE_NUM}/${NEW_DATE_NUM}/g" namelist_cfg
  fi
  echo " "
  echo " "
  echo " "
  echo " "
  echo " ... a wild Totoro appeared and blocked your resubmission!"
  echo "          ,--'''',--.__,---[],-------._                                  "
  echo "        ,'    __,'             \         \--''''''==;-                  "
  echo "      ,' _,-'  '/---.___      \       ___\   ,-'','                    "
  echo "    /,-'       / ;. ,.--'-.__\  _,-'' ,| ',',   /                      "
  echo "   /''''''-._/,-|:\       []\,' '''-/:;-. '. /                          "
  echo "              '  ;:::      ||       /:,;  '-.\                          "
  echo "                =.,'__,---||-.____',.=                                   "
  echo "                =(:\_     ||__    ):)=                                   "
  echo "               ,'::::'----||::'--':::'._                                 "
  echo "             ,':::::::::::||::::::::::::'.                               "
  echo "    .__     ;:::.-.:::::__||___:::::.-.:::\     __,                      "
  echo "       '''-;:::( O ):::>_|| _<::::( O ):::-'''                          "
```

(continues on next page)

(continued from previous page)

```
  echo "   =======;::::::'-':::::::||':::::::'-':::::\=======                  "
  echo "    ,--'';:::_____________||_____________:::::''----.          , ,    "
  echo "         ; ::'._(    |    |||     |   )_,':::::\_,,,,,,,,,,,____/,'_,  "
  echo "       ,;    :::'--._|____[]|_____|_.-'::::::::::::::::::::::::);_    "
  echo "      ;/ /      :::::::::,||,:::::::::::::::::::::::::::::::::/      "
  echo "     /; ''''''----------/,'/,__,,,,,____:::::::::::::::::::,'        "
  echo "     ;/                :);/|_;| ,--.. . '''-.:::::::::::::_,'         "
  echo "    /;                :::):__,'//''\\. ,--.. \:::,::::_,'             "
  echo "   ;/              :::::/ . . . . . . //''\\. \::':__,'               "
  echo "   ;/           :::::::,' . . . . . . . . . . .:'::\                  "
  echo "   ';       :::::::__,'. ,--.. . .,--. . . . . .:'::'                 "
  echo "   ';   __,..--'''-. . //''\\. .//''\\ . ,--.. :':::'                 "
  echo "   ;    /  \\ .//''\\ . . . . . . . . . //''\\. :'::'                 "
  echo "   ;   /        . . . . . . . . . . . . . . . .:'::'                 "
  echo "   ;   (           . . . . . . . . . . . . . . ;:::'                 "
  echo "   ,:  ;,            . . . . . . . . . . . . .;'::'                 "
  echo "   ,:  ;,             . . . . . . . . . . . .;'::'                  "
  echo "   ,:   ;,             . . . . . . . . . . . ;'::;'                 "
  echo "     :   ;              . . . . . . . . . . .,':::;                 "
  echo "      :   '.           . . . . . . . . .. . .,':::;'                "
  echo "       :    '.        . . . . . . . . . . . .;:::::'                "
  echo "        '.    '-.    . . . . . . . . . . . .,-':::::;               "
  echo "          ':_     ''--.._____________..--'':::::;''                 "
  echo "             '._::,.:,.:,:_ctr_:,:,.::,.:_;''                       "
  echo "_________________''\/'\/\/''''''\/'\/''\/'_____________________________"

else
# WARNING: this assumes that OLD_DATE_NUM is the only number within the file, which␣
↪should
  #          really be true
  sed -i "s/${OLD_DATE_NUM}/${NEW_DATE_NUM}/g" namelist_cfg

  echo "grabbed time stamp ${NEW_DATE_NUM} smaller than threshold ${THRESH},␣
↪resubmitting..."
  echo "grabbed time stamp ${NEW_DATE_NUM} smaller than threshold ${THRESH},␣
↪resubmitting..."
  echo "grabbed time stamp ${NEW_DATE_NUM} smaller than threshold ${THRESH},␣
↪resubmitting..."
  echo "grabbed time stamp ${NEW_DATE_NUM} smaller than threshold ${THRESH},␣
↪resubmitting..."
  echo " "
  echo "OK: ...and here is Christopher resbumitting the job for you....."
  echo "                  ,-.____,-.          "
  echo "                 /   ..   \          "
  echo "                /_       _\          "
  echo "               |'o'     'o'|         "
  echo "              / ___________ \         "
  echo "             , ,'    '--'    '. .     "
  echo "           _| |               | |_    "
  echo "         /  ' '               ' '  \  "
  echo "        (    ',',__________.','    ) "
  echo "         \_     ' ._______, '     _/  "
```

(continues on next page)

(continued from previous page)

```
  echo "              |                   |     "
  echo "              |    ,-.    ,-.     |     "
  echo "               \       ).,(      /      "
  echo "         gpyy   \___/    \___/         "
  sbatch submit_nemo

fi

exit
```

The output recombination steps have bene commented out because ARC does have parallel NetCDF4 and so the `one_file` option in `field_def_nemo.xml` already takes care of the outputs.

1.6 HKUST HPC2 compilation

The build uses NEMO 3.7/4.0 + XIOS 2.0 as the example. For installing other versions, extrapolate from the other notes.

HKUST HPC2 is a cluster with SLURM. Modules are loaded on an per basis via sourcing some shell scripts. The following is going to use `gcc` and `openmpi`, but in theory the corresponding intel compilers should work too (not tested):

```
source /usr/local/setup/gcc-g++-4.9.2.sh
source /usr/local/setup/openmpi-2.0.0.sh
```

The notes are **(psuedo)-chronological** (complete with errors) rather than the final product to highlight some pitfalls and workarounds to do with HDF5 and NetCDF4 compatibility (the system itself does not have parallel HDF5 or NetCDF4 and it was a mystery which compiler the libraries were built with).

1.6.1 XIOS (1st try that doesn't quite work)

> **Warning:** Doing whatever is detailed here in this subsection will get XIOS compiled, but then it turns out when compiling NEMO that the system NetCDF4 is incompatible with the chosen compiler (I still have no idea which compiler was used for the system NetCDF4). The final working solution is to compile (a much more up-to-date) HDF5 and NetCDF4 separately; this means the final `arch-HKUST_HPC2.env` will be different.

I did the usual things of downloading XIOS and copying the arch files in

```
cd $PI_HOME # <--- this is the "work" directory (which is generically not ~/)
mkdir XIOS
cd XIOS
svn checkout -r 1322 http://forge.ipsl.jussieu.fr/ioserver/svn/XIOS/trunk xios-2.0
cd xios2.0/arch
cp arch-GCC_LINUX.env arch-HKUST_HPC2.env
cp arch-GCC_LINUX.fcm arch-HKUST_HPC2.fcm
cp arch-GCC_LINUX.path arch-HKUST_HPC2.path
```

HDF5 and NetCDF4 is not included in a load so it is there by default. Doing for example `locate libnetcdf` tells me that I should have the following:

```
# arch-HKUST_HPC2.env

export HDF5_INC_DIR=/usr/include
export HDF5_LIB_DIR=/usr/lib64

export NETCDF_INC_DIR=/usr/include
export NETCDF_LIB_DIR=/usr/lib64
```

Following this I did

```
export LD_LIBRARY_PATH="/usr/lib64:$LD_LIBRARY_PATH"
```

as it seems to help the programs find the libraries.

I must have done something a bit weird to the `arch-HKUST_HPC2.path`:

```
NETCDF_INCDIR="-I$NETCDF_INC_DIR"
#NETCDF_LIBDIR='-Wl,"--allow-multiple-definition" -Wl,"-Bstatic" -L$NETCDF_LIB_DIR'
NETCDF_LIBDIR='-Wl,"--allow-multiple-definition" -L$NETCDF_LIB_DIR'
NETCDF_LIB="-lnetcdff -lnetcdf"

HDF5_LIBDIR="-L$HDF5_LIB_DIR"
HDF5_LIB="-lhdf5_hl -lhdf5 -lz"
```

Not sure where I got the `-Bstatic` flag from initially (maybe from the ARCHER compilation). If that flag is there when doing the compiling then I get the error

```
### ERROR ###
linker error: ld cannot locate lnetcdf etc.
```

but doing something like `ld [-L/usr/lib64] -lnetcdf --verbose` or using whatever the `ld` is actually called because of the modified `$PATH` clearly shows success. The same happens when the intel compilers are used. Anyway, using the following (the system had `gmake` so I left it; `make` should work too)

```
# arch-HKUST_HPC2.fcm

###############################################################################
###################                Projet XIOS               ##################
###############################################################################

%CCOMPILER      mpicc
%FCOMPILER      mpif90
%LINKER         mpif90

%BASE_CFLAGS    -ansi -w
%PROD_CFLAGS    -O3 -DBOOST_DISABLE_ASSERTS
%DEV_CFLAGS     -g -O2
%DEBUG_CFLAGS   -g

%BASE_FFLAGS    -D__NONE__ -ffree-line-length-none
%PROD_FFLAGS    -O3
%DEV_FFLAGS     -g -O2
%DEBUG_FFLAGS   -g
```

(continues on next page)

(continued from previous page)

```
%BASE_INC       -D__NONE__
%BASE_LD        -lstdc++

%CPP            cpp
%FPP            cpp -P
%MAKE           gmake
```

followed by

```
cd ../
[CPPFLAGS=-I/usr/include LDFLAGS=-L/usr/lib64] ./make_xios --full --prod --arch HKUST_
↪HPC2 -j4 |& tee compile_log.txt
```

seems to do the job. I think I did go into `bld.cfg` and changed `src_netcdf` to `src_netcdf4` for safety; don't remember needing this in ARCHER (did need it when doing a local compilation).

1.6.2 NEMO (1st try that doesn't quite work)

Warning: Again this doesn't quite work because of NetCDF4 Fortran compiler compatibility. The final working `arch-HKUST_HPC2.fcm` has a modified `%NCDF_INC` and `%NCDF_LIB`.

As advertised, when doing the following

```
cd $PI_HOME
mkdir NEMO
cd NEMO
svn checkout -r 8666 http://forge.ipsl.jussieu.fr/nemo/svn/NEMO/trunk nemo3.7-8666
cd nemo3.7-8666/NEMOGCM/ARCH
cp OLD/arch-gfortran_linux.fcm ./arch-HKUST_HPC2.fcm
```

using

```
# arch-HKUST_HPC2.fcm
# generic gfortran compiler options for linux
# NCDF_INC    netcdf include file
# NCDF_LIB    netcdf library
# FC          Fortran compiler command
# FCFLAGS     Fortran compiler flags
# FFLAGS      Fortran 77 compiler flags
# LD          linker
# LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
# FPPFLAGS    pre-processing flags
# AR          assembler
# ARFLAGS     assembler flags
# MK          make
# USER_INC    additional include files for the compiler,  e.g. -I<include dir>
# USER_LIB    additional libraries to pass to the linker, e.g. -l<library>

%XIOS_HOME           $PI_HOME/XIOS/xios-2.0
```

(continues on next page)

(continued from previous page)

```
%CPP                 cpp
%CPPFLAGS            -P -traditional

%XIOS_INC            -I%XIOS_HOME/inc
%XIOS_LIB            -L%XIOS_HOME/lib -lxios

%NCDF_INC            -I/usr/include
%NCDF_LIB            -L/usr/lib64 -lnetcdf -lnetcdff -lstdc++
%FC                  mpif90
%FCFLAGS             -fdefault-real-8 -O3 -funroll-all-loops -fcray-pointer -cpp -ffree-
↪line-length-none
%FFLAGS              %FCFLAGS
%LD                  %FC
%LDFLAGS
%FPPFLAGS            -P -C -traditional
%AR                  ar
%ARFLAGS             -rs
%MK                  make
%USER_INC            %XIOS_INC %NCDF_INC
%USER_LIB            %XIOS_LIB %NCDF_LIB
```

When building with

```
cd ../CONFIG
./makenemo -r GYRE_PISCES -n GYRE_testing -m HKUST_HPC2 -j0
nano GYRE_testing/cpp_GYRE_testing.fcm # (have key_top -> key_nosignedzero)
./makenemo -n GYRE_tesitng -m HKUST_HPC2 -j4
```

throws up the error that NetCDF4 being called was built with a different gfortran compiler. So the workaround here is build the dependencies separately...

1.6.3 zlib, HDF5 and NetCDF4

I have not figured out how to get the parallel builds of HDF5 and NetCDF4 done successfully. Without it NEMO still works fine it just means each processor spits out the data associated with the tile it is assigned to: the `one_file` option in `file_def_nemo.xml` doesn't work without parallel NetCDF4 and only `multiple_file` is allowed (it will crash the first time step it tries to write). The workaround here is to at the post-processing stage rely on the NEMO `TOOLS/REBUILD_NEMO` to recombine the files if required.

I built everything as follows (see *here* for more details on the commands maybe):

> **Warning:** `LD_LIBRARY_FLAG` definitely does not point to `/usr/lib64` now, though I don't remember if I strictly needed to set it to `$PI_HOME/custom_libs/lib`

```
### initialise
cd $PI_HOME
mkdir custom_libs
cd custom_libs
mkdir sources
cd sources
```

(continues on next page)

(continued from previous page)

```
# zlib
wget http://www.zlib.net/zlib-1.2.11.tar.gz
tar -xvzf $BD/source/zlib-1.2.11.tar.gz
cd zlib-1.2.11
CFLAGS=-fPIC ./configure --prefix=$PI_HOME/custom_libs # -fPIC for shared libraries
make -j 4
make check install

# HDF5
cd $PI_HOME/custom_libs/sources
wget https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.8/hdf5-1.8.19/src/hdf5-1.8.19.
↪tar.gz
tar -xvzf $BD/source/hdf5-1.8.19.tar.gz
cd hdf5-1.8.19
CPPFLAGS=-I$PI_HOME/custom_libs/include LDFLAGS=-L$PI_HOME/custom_libs/lib \
  CFLAGS=-fPIC ./configure --enable-shared --enable-fortran --prefix=$PI_HOME/custom_libs
make -j 4
make check install # <---- this step takes a while

# NetCDF (C)
cd $PI_HOME/custom_libs/sources
wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4.4.1.1.tar.gz
tar -xvzf $BD/source/netcdf-4.4.1.1.tar.gz
cd netcdf-4.4.1.1
CPPFLAGS=-I$PI_HOME/custom_libs/include LDFLAGS=-L$PI_HOME/custom_libs/lib \
  ./configure --enable-netcdf4 --enable-shared --prefix=$PI_HOME/custom_libs
make -j 4
make check install # <---- this step takes a while

# NetCDF (Fortran)
cd $PI_HOME/custom_libs/sources
wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-fortran-4.4.4.tar.gz
tar -xvzf $BD/source/netcdf-fortran-4.4.4.tar.gz
cd netcdf-fortran-4.4.4
CPPFLAGS=-I$PI_HOME/custom_libs/include LDFLAGS=-L$PI_HOME/custom_libs/lib \
  ./configure --enable-shared --prefix=$PI_HOME/custom_libs
make -j 4
make check install
```

My written notes says I made sure `LD_LIBRARY_PATH` pointed to `$PI_HOME/custom_libs/libs` for the NetCDF4-fortran `./configure` part.

1.6.4 Building XIOS and NEMO again

I rebuilt XIOS after changing `arch-HKUST_HPC2.env` to (probably added to `LD_LIBRARY_PATH`):

```
# arch-HKUST_HPC2.env

export HDF5_INC_DIR=$PI_HOME/custom_libs/include
export HDF5_LIB_DIR=$PI_HOME/custom_libs/lib

export NETCDF_INC_DIR=$PI_HOME/custom_libs/include
export NETCDF_LIB_DIR=$PI_HOME/custom_libs/lib
```

For the NEMO part, `arch-HKUST_HPC2.fcm` now has the following:

```
%NCDF_INC            -I/$PI_HOME/custom_libs/include
%NCDF_LIB            -L$PI_HOME/custom_libs/lib -lnetcdf -lnetcdff -lstdc++
```

Then finally everything works. I'm going to make use of the NEMO `TOOLS/REBUILD_NEMO` to have a single NetCDF file so I additionally do the following (starting from the `CONFIG` folder):

```
cd ../TOOLS
./maketools -n REBUILD_NEMO -m HKUST_HPC2
```

which results in a `TOOLS/REBUILD_NEMO/rebuild_nemo.exe` that I am going to use in my post-processing script later.

1.6.5 Running NEMO on the HPC2

The system uses SLURM and the key commands are

- `sbatch [submit_nemo]`: submits the job detailed in `submit_nemo` (see below)
- `scancel [job ID]`: cancel the job
- `sinfo`: check status of queues available
- `squeue -u $USER`: check job info for `$USER`

`sbatch` could be used with arguments but I am going to have everything within `submit_nemo` itself. The generic one I use (based on the one given on the NOCL page) is as follows (I have some ASCII art in there because I got bored at some point):

```
#!/bin/bash

# NOTE: Lines starting with "#SBATCH" are valid SLURM commands or statements,
#       while those starting with "#" and "##SBATCH" are comments.  Uncomment
#       "##SBATCH" line means to remove one # and start with #SBATCH to be a
#       SLURM command or statement.

#=============================================================
# DEFINE SOME JUNK FOR THE SUBMISSION (??? make this more flexible with e.g. queues?)
#=============================================================

#SBATCH -J gyre04       # job name
#SBATCH -o stdouterr    # output and error file name
#SBATCH -n 24           # total number of mpi tasks requested
```

(continues on next page)

(continued from previous page)

```
#SBATCH -N 1            # total number of nodes requested
#SBATCH -p ssci         # queue (partition) -- standard, development, etc.
#SBATCH -t 12:00:00     # maximum runtime

# Enable email notificaitons when job begins and ends, uncomment if you need it
##SBATCH --mail-user=user_name@ust.hk #Update your email address
##SBATCH --mail-type=begin
##SBATCH --mail-type=end

# Setup runtime environment if necessary
# For example, setup MPI environment
source /home/jclmak/nemo_env.sh
# or you can source ~/.bashrc or ~/.bash_profile

#================================================================
# LAUNCH JOB
#================================================================

echo " _ __   ___ _ __ ___   ___          "
echo "| '_ \ / _ \ '_ ' _ \ / _ \         "
echo "| | | |  __/ | | | | | (_) |        "
echo "|_| |_|\___|_| |_| |_|\___/  v3.7   "

# Go to the job submission directory and run your application
cd $PI_HOME/NEMO/nemo3.7-8666/NEMOGCM/CONFIG/GYRE_testing/EXP00/
mpirun -n 24 ./opa

#================================================================
# POSTPROCESSING
#================================================================

# kills the daisy chain if there are errors

if grep -q 'E R R O R' ocean.output ; then

  echo "E R R O R found, exiting..."
  echo "  ___ _ __ _ __ ___  _ __   "
  echo " / _ \ '__| '__/ _ \| '__|  "
  echo "|  __/ |  | | | (_) | |     "
  echo " \___|_|  |_|  \___/|_|     "
  echo "check out ocean.output or stdouterr to see what the deal is "

  exit
else
  echo "going into postprocessing stage..."
  # cleans up files, makes restarts, moves files, resubmits this pbs

  bash ./postprocess.sh >& cleanup.log
  exit
fi
```

Here because I am not using `xios_server.exe` I don't strictly need the `-n 24` after `mpirun` (it will then just use however many cores that's given in `#SBATCH -n`). Maybe see the *Oxford ARC* one to see how it might work when

`xios_server.exe` is run alongside NEMO to do the I/O .

The following post-processing script requires a few prepping (I make no apologies for the bad code and the script being fickle; feel free to modify as you see fit):

- copying the `nn_date0` line into `namelist_cfg` from say `namelist_ref` if it doesn't exist already, because the time-stamps are modified by modifying `nn_date0`
- do a search in `namelist_cfg` and make sure there is only ever one mention of `nn_date0` (otherwise it grabs the wrong lines)
- `nn_date0` should not begin with zeros (e.g. `10101` rather than `010101` in `yymmdd`)
- in the experiment folder, do `mkdir RESTARTS OUTPUTS` (otherwise there is no folder to copy into)

The `postprocess.sh` I cooked up is here:

```
#!/bin/bash
#! postprocess.sh
#! Script to clean up the NEMO outputs

export BASE_DIR=$PI_HOME/NEMO/nemo3.7-8666/NEMOGCM/
export MODEL=GYRE
export NUM_CPU=24

# time-stamp increment, yymmdd
export DATE_INC=100000

# when to stop the daisy chaining, yymmdd
export THRESH=10

# error catching (only when restart files etc cannot be copied or made)
export ERR_CATCH=0

#########################################################
# 0) recombine files to one netcdf (restarts and/or outputs)
# restarts: extract the restart file time-step stamp
#              based on the *0000.nc restart which should (!) always exist
#           rebuild the restart file in the submission directory
# outputs:  put them in manually and just do a grab
#           this assumes only files at the current time-stamp is there,
#              otherwise it will bug out as it grabs wrong files
#########################################################

# restart files
export RES_TIMESTAMP=$(echo $(ls -d ${MODEL}_*_restart_0000.nc) | awk -F _ '{print $2 }')

$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${RES_TIMESTAMP}_restart $NUM_CPU
if (($? > 0)); then
  ERR_CATCH=$((ERR_CATCH + 1))
  echo "  ERR: making the restart file in the folder"
fi
##$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${RES_TIMESTAMP}_restart_ice $NUM_
↪CPU

# output files (assumes a grid_T always exists)
```

(continues on next page)

(continued from previous page)

```
export OUT_FREQ=$(echo $(ls -d ${MODEL}_*_grid_T_0000.nc) | awk -F _ '{print $2 }')
export OUT_START=$(echo $(ls -d ${MODEL}_*_grid_T_0000.nc) | awk -F _ '{print $3 }')
export OUT_END=$(echo $(ls -d ${MODEL}_*_grid_T_0000.nc) | awk -F _ '{print $4 }')

$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${OUT_FREQ}_${OUT_START}_${OUT_END}_
↪grid_T $NUM_CPU
$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${OUT_FREQ}_${OUT_START}_${OUT_END}_
↪grid_U $NUM_CPU
$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${OUT_FREQ}_${OUT_START}_${OUT_END}_
↪grid_V $NUM_CPU
$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${OUT_FREQ}_${OUT_START}_${OUT_END}_
↪grid_W $NUM_CPU

# add more things in here if output freqs are different etc

#########################################################
# 1) pull out some variables to modify namelist file
#########################################################

# pull the number out
# add the increment to it for new date
# subtract appropriately to get the date stamp
#   (e.g. 110101 - 8871 = 101230) and bulk out zeros

export OLD_DATE_STR=$(grep -ri "nn_date0" namelist_cfg)
export OLD_DATE_NUM=$(echo ${OLD_DATE_STR} | sed -e 's/[^0-9 ]//g' | awk '{print $NF}')
export NEW_DATE_NUM=$((OLD_DATE_NUM + DATE_INC))

# 8871 for 30 days a month (so the RES_STAMP=yyyy1230)
# otherwise do 8870        (so the RES_STAMP=yyyy1231)
# do something else for other time units
export RES_STAMP=$(printf %08d $((NEW_DATE_NUM - 8871)))

#########################################################
# 2) move files around and tidy up
#########################################################

cp -pv ${MODEL}_${RES_TIMESTAMP}_restart.nc ./RESTARTS/${MODEL}_${RES_STAMP}_restart.nc
cp -pv ./output.namelist.dyn ./OUTPUTS/output.namelist.dyn.${RES_STAMP}
#cp -pv ${MODEL}_${RES_TIMESTAMP}_restart_ice.nc ./RESTARTS/${MODEL}_${RES_STAMP}_
↪restart_ice.nc
#cp -pv ./output.namelist.ice ./OUTPUTS/output.namelist.ice.${RES_STAMP}
cp -pv ./ocean.output ./OUTPUTS/ocean.output.${RES_STAMP}
cp -pv ./solver.stat ./OUTPUTS/solver.stat.${RES_STAMP}
cp -pv ./stdouterr ./OUTPUTS/stdouterr.${RES_STAMP}
cp -pv ./namelist_cfg ./OUTPUTS/namelist_cfg.${RES_STAMP}

#cp -pv ./volume_transport ./OUTPUTS/volume_transport.${RES_STAMP}
#cp -pv ./salt_transport ./OUTPUTS/salt_transport.${RES_STAMP}
#cp -pv ./heat_transport ./OUTPUTS/heat_transport.${RES_STAMP}

rm -v ${MODEL}_${RES_TIMESTAMP}_restart*
```

(continues on next page)

(continued from previous page)

```
rm -v restart.nc
#rm -v restart_ice.nc
rm -v ${MODEL}_*_????.nc
mv ${MODEL}*.nc ./OUTPUTS

cp -pv RESTARTS/${MODEL}_${RES_STAMP}_restart.nc ./restart.nc
if (($? > 0)); then
  ERR_CATCH=$((ERR_CATCH + 1))
  echo "  ERR: copying restart file into folder"
fi

#cp -pv RESTARTS/${MODEL}_${RES_STAMP}_restart_ice.nc ./restart_ice.nc
#if (($? > 0)); then
#  ERR_CATCH=$((ERR_CATCH + 1))
#  echo "  ERR: copying restart_ice file into folder"
#fi

############################################################
# 3) if all good, then modify namelist_cfg and resbumit
############################################################

if (($ERR_CATCH > 0)) || ((${NEW_DATE_NUM} > $THRESH)); then
  if (($ERR_CATCH > 0)); then
    echo " "
    echo " "
    echo " "
    echo "ERR: caught a non-zero exit status, check cleanup.log for what the deal was"
    echo "ERR: caught a non-zero exit status, check cleanup.log for what the deal was"
  else
    echo "OK: grabbed time stamp ${NEW_DATE_NUM} larger than threshold ${THRESH},␣
↪breaking..."
    echo "OK: grabbed time stamp ${NEW_DATE_NUM} larger than threshold ${THRESH},␣
↪breaking..."
    # WARNING: this assumes that OLD_DATE_NUM is the only number within the file, which␣
↪should
    #          really be true
    sed -i "s/${OLD_DATE_NUM}/${NEW_DATE_NUM}/g" namelist_cfg
  fi
  echo " "
  echo " "
  echo " "
  echo " "
  echo " ... a wild Totoro appeared and blocked your resubmission!"
  echo "          ,--'''',--.__,---[],-------._                                  "
  echo "        ,'   __,'            \         \--''''''==;-                    "
  echo "      ,' _,-'  '/---.___      \       ___\   ,-'','                     "
  echo "    /,-'      / ;. ,.--'-.__\  _,-'' ,| ',',   /                       "
  echo "   /''''''-._/,-|:\       []\,' '''-/:;-. '. /                          "
  echo "              '  ;:::      ||       /:,;  '-.\                          "
  echo "                =.,'__,---||-.____',.=                                   "
  echo "                =(:\_     ||__    ):)=                                  "
  echo "               ,'::::'----||::'--':::'._                                 "
```

(continues on next page)

(continued from previous page)

```
  echo "             ,':::::::::::||::::::::::::'.                                "
  echo "    .__     ;::::.-.:::::__||___:::::.-.:::\     __,                     "
  echo "       '''-;::::( O ):::::>_|| _<::::( O ):::::-'''                      "
  echo "   =======;::::::'-':::::::||':::::::'-':::::\=======                    "
  echo "    ,--'';:::_____________||_____________:::::''----.          , ,       "
  echo "         ; ::'._(    |    |||     |   )_,':::::\_,,,,,,,,,,,_____/,'_,  "
  echo "       ,;    :::'--._|____[]|_____|_.-'::::::::::::::::::::::::::);_    "
  echo "      ;/ /      :::::::::,||,:::::::::::::::::::::::::::::::::::/       "
  echo "     /; ''''''----------/,'/,__,,,,,____:::::::::::::::::::::,'         "
  echo "     ;/                :);/|_;| ,--.. . '''-.:::::::::::::_,'           "
  echo "    /;               :::):__,'//''\\. ,--.. \:::,:::::_,'               "
  echo "   ;/              :::::/ . . . . . //''\\. \::':__,'                   "
  echo "   ;/           :::::::,' . . . . . . . . . .:'::\                     "
  echo "   ';      :::::::__,'. ,--.. . .,--. . . . .:'::'                     "
  echo "   ';   __,..--'''-. . //''\\. .//''\\ . ,--.. :':::'                   "
  echo "   ;    /  \\ .//''\\ . . . . . . . . //''\\. :'::'                    "
  echo "   ;   /       . . . . . . . . . . . . . . .:'::'                     "
  echo "   ;   (          . . . . . . . . . . . . . . ;:::'                   "
  echo "   ,:  ;,           . . . . . . . . . . . . ;':::'                    "
  echo "   ,:  ;,            . . . . . . . . . . . .;':::'                    "
  echo "   ,:   ;,            . . . . . . . . . . . ;'::;'                    "
  echo "     :   ;            . . . . . . . . . . . ,':::;                    "
  echo "      :   '.          . . . . . . . . .. . . .,':::;'                 "
  echo "       :    '.       . . . . . . . . . . . . ;::::;'                  "
  echo "        '.    '-.   . . . . . . . . . . . ,-':::::;                   "
  echo "          ':_     ''--..__________..--'':::::;''                      "
  echo "             '._::,.:,.:,:_ctr_:,:,.::,.:_;''                         "
  echo "_________________''\/'\/\/''''''\/'\/''\/'_____________________________"

else
# WARNING: this assumes that OLD_DATE_NUM is the only number within the file, which␣
↪should
  #          really be true
  sed -i "s/${OLD_DATE_NUM}/${NEW_DATE_NUM}/g" namelist_cfg

  echo "grabbed time stamp ${NEW_DATE_NUM} smaller than threshold ${THRESH},␣
↪resubmitting..."
  echo "grabbed time stamp ${NEW_DATE_NUM} smaller than threshold ${THRESH},␣
↪resubmitting..."
  echo "grabbed time stamp ${NEW_DATE_NUM} smaller than threshold ${THRESH},␣
↪resubmitting..."
  echo "grabbed time stamp ${NEW_DATE_NUM} smaller than threshold ${THRESH},␣
↪resubmitting..."
  echo " "
  echo "OK: ...and here is Christopher resbumitting the job for you....."
  echo "                  ,-.____,-.          "
  echo "                  /   ..   \          "
  echo "                 /_        _\         "
  echo "                |'o'      'o'|        "
  echo "               / ____________ \        "
  echo "             , ,'    '--'    '. .     "
  echo "            _| |              | |_    "
```

(continues on next page)

(continued from previous page)

```
  echo "           /  ' '               ' '  \  "
  echo "          (    ',',__________.','    ) "
  echo "           \_     ' ._______, '     _/  "
  echo "              |                   |     "
  echo "              |    ,-.    ,-.     |     "
  echo "               \       ).,(      /      "
  echo "          gpyy  \___/    \___/         "
  sbatch submit_nemo

fi

exit
```

A chunk of the output recombination procedures are not required if the `one_file` option in `field_def_nemo.xml` is enabled and possible (requires parallel NetCDF4 which I didn't bother building here).

1.7 HKUST HPC3 compilation

The build uses NEMO 3.7/4.0 + XIOS 2.0 as the example. For installing other versions, extrapolate from the other notes.

HKUST HPC3 is a cluster with SLURM. The usual `module load/swap/purge/unload/list/avail` works here, and the system so far is built by default with `gcc8.4` and the relevant netCDF4 and HDF5 libraries (serial and parallel version).

1.7.1 Compilers

Note: HPC3 now has the gcc5.4 compilers at `module swap gnu/8.4.0 gnu/5.4.0`. MPI bindings not available so those still need to be built.

So the first problem is compilers for XIOS. As far as I can tell (I am happy to be wrong), as of writing, XIOS doesn't play well with `gcc` versions above 6 and so using the system compilers will fail, and indeed building XIOS as per usual hits the c++ standard and some routine naming errors (my understanding is that the newer versions of gcc are more strict with naming). So I decided to build the compilers myself (and with it all the other libraries just for safety). See the *packages* page.

After a few hours (it takes that long for a bootstrap build) I have `gcc5.4` in `/scratch/PI/jclmak/custom_libs/gcc5.4/`. I proceeded to relogin, unload `gcc8.4` and building the libraries into the same target folder for safety (needed to build `m4`). I have a specific environment file that includes

```
export LD_LIBRARY_PATH="/scratch/PI/jclmak/custom_libs/gcc5.4/lib:$LD_LIBRARY_PATH"
```

1.7.2 XIOS

I did the usual things of downloading XIOS and copying the arch files in

```
cd $PI_HOME # <--- this is the "work" directory (which is generically not ~/)
mkdir XIOS
cd XIOS
svn checkout -r 1322 http://forge.ipsl.jussieu.fr/ioserver/svn/XIOS/trunk xios-2.0
cd xios2.0/arch
cp arch-GCC_LINUX.env arch-HKUST_HPC3.env
cp arch-GCC_LINUX.fcm arch-HKUST_HPC3.fcm
cp arch-GCC_LINUX.path arch-HKUST_HPC3.path
```

Since I built all the libraries separately the arch files look like the following:

```
# arch-HKUST_HPC3.env

export HDF5_INC_DIR=/scratch/PI/jclmak/custom_libs/gcc5.4/include
export HDF5_LIB_DIR=/scratch/PI/jclmak/custom_libs/gcc5.4/lib

export NETCDF_INC_DIR=/scratch/PI/jclmak/custom_libs/gcc5.4/include
export NETCDF_LIB_DIR=/scratch/PI/jclmak/custom_libs/gcc5.4/lib
```

```
# arch-HKUST_HPC3.path

NETCDF_INCDIR="-I$NETCDF_INC_DIR"
NETCDF_LIBDIR='-Wl,"--allow-multiple-definition" -L$NETCDF_LIB_DIR'
NETCDF_LIB="-lnetcdff -lnetcdf"

HDF5_LIBDIR="-L$HDF5_LIB_DIR"
HDF5_LIB="-lhdf5_hl -lhdf5 -lz"
```

```
# arch-HKUST_HPC3.fcm

###############################################################################
###################                Projet XIOS               ##################
###############################################################################

%CCOMPILER      mpicc
%FCOMPILER      mpif90
%LINKER         mpif90

%BASE_CFLAGS    -ansi -w -D_GLIBCXX_USE_CXX11_ABI=0
%PROD_CFLAGS    -O3 -DBOOST_DISABLE_ASSERTS
%DEV_CFLAGS     -g -O2
%DEBUG_CFLAGS   -g

%BASE_FFLAGS    -D__NONE__ -ffree-line-length-none
%PROD_FFLAGS    -O3
%DEV_FFLAGS     -g -O2
%DEBUG_FFLAGS   -g

%BASE_INC       -D__NONE__
```

(continues on next page)

(continued from previous page)

```
%BASE_LD        -lstdc++

%CPP            cpp
%FPP            cpp -P
%MAKE           make
```

The `-D_GLIBCXX_USE_CXX11_ABI=0` is needed because we are using `gcc5.4`. Then I ran

```
cd ../
[CPPFLAGS=-I/usr/include LDFLAGS=-L/usr/lib64] ./make_xios --full --prod --arch HKUST_
↪HPC3 |& tee compile_log.txt
```

I think I did go into `bld.cfg` and changed `src_netcdf` to `src_netcdf4` for safety.

1.7.3 NEMO

Load subversion with `module load subversion` and do

```
cd $PI_HOME
mkdir NEMO
cd NEMO
svn checkout -r 8666 http://forge.ipsl.jussieu.fr/nemo/svn/NEMO/trunk nemo3.7-8666
cd nemo3.7-8666/NEMOGCM/ARCH
cp OLD/arch-gfortran_linux.fcm ./arch-HKUST_HPC3.fcm
```

and have

```
# arch-HKUST_HPC3.fcm
# generic gfortran compiler options for linux
# NCDF_INC    netcdf include file
# NCDF_LIB    netcdf library
# FC          Fortran compiler command
# FCFLAGS     Fortran compiler flags
# FFLAGS      Fortran 77 compiler flags
# LD          linker
# LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
# FPPFLAGS    pre-processing flags
# AR          assembler
# ARFLAGS     assembler flags
# MK          make
# USER_INC    additional include files for the compiler,  e.g. -I<include dir>
# USER_LIB    additional libraries to pass to the linker, e.g. -l<library>

%XIOS_HOME           /scratch/PI/jclmak/XIOS/xios-2.0

%CPP                 cpp
%CPPFLAGS            -P -traditional

%XIOS_INC            -I%XIOS_HOME/inc
%XIOS_LIB            -L%XIOS_HOME/lib -lxios

%NCDF_INC            -I/scratch/PI/jclmak/custom_libs/gcc5.4/include
```

(continues on next page)

(continued from previous page)

```
%NCDF_LIB            -L/scratch/PI/jclmak/custom_libs/gcc5.4/lib -lnetcdf -lnetcdff -
↪lstdc++
%FC                  mpif90
%FCFLAGS             -fdefault-real-8 -O3 -funroll-all-loops -fcray-pointer -cpp -ffree-
↪line-length-none
%FFLAGS              %FCFLAGS
%LD                  %FC
%LDFLAGS
%FPPFLAGS            -P -C -traditional
%AR                  ar
%ARFLAGS             -rs
%MK                  make
%USER_INC            %XIOS_INC %NCDF_INC
%USER_LIB            %XIOS_LIB %NCDF_LIB
```

and build with

```
cd ../CONFIG
./makenemo -r GYRE_PISCES -n GYRE_testing -m HKUST_HPC3 -j0
nano GYRE_testing/cpp_GYRE_testing.fcm # (have key_top -> key_nosignedzero)
./makenemo -n GYRE_tesitng -m HKUST_HPC3 -j4
```

I'm going to make use of the NEMO `TOOLS/REBUILD_NEMO` to have a single NetCDF file so I additionally do the following (starting from the `CONFIG` folder):

```
cd ../TOOLS
./maketools -n REBUILD_NEMO -m HKUST_HPC2
```

which results in a `TOOLS/REBUILD_NEMO/rebuild_nemo.exe` that I am going to use in my post-processing script later.

1.7.4 Running NEMO on the HPC2

The system uses SLURM and the key commands are

- `sbatch [submit_nemo]`: submits the job detailed in `submit_nemo` (see below)
- `scancel [job ID]`: cancel the job
- `sinfo`: check status of queues available
- `squeue -u $USER`: check job info for `$USER`

`sbatch` could be used with arguments but I am going to have everything within `submit_nemo` itself. The generic one I use (based on the one given on the NOCL page) is as follows (I have some ASCII art in there because I got bored at some point):

```
#!/bin/bash

# NOTE: Lines starting with "#SBATCH" are valid SLURM commands or statements,
#       while those starting with "#" and "##SBATCH" are comments.  Uncomment
#       "##SBATCH" line means to remove one # and start with #SBATCH to be a
#       SLURM command or statement.
```

(continues on next page)

(continued from previous page)

```
#=================================================================
# DEFINE SOME JUNK FOR THE SUBMISSION (??? make this more flexible with e.g. queues?)
#=================================================================

#SBATCH -J gyre04       # job name
#SBATCH -o stdouterr    # output and error file name
#SBATCH -n 40           # total number of mpi tasks requested
#SBATCH -N 1            # total number of nodes requested
#SBATCH -p oces         # queue (partition) -- standard, development, etc.
#SBATCH -t 12:00:00     # maximum runtime

# Enable email notificaitons when job begins and ends, uncomment if you need it
##SBATCH --mail-user=user_name@ust.hk #Update your email address
##SBATCH --mail-type=begin
##SBATCH --mail-type=end

# Setup runtime environment if necessary
# For example, setup MPI environment
source /home/jclmak/nemo_env.sh
# or you can source ~/.bashrc or ~/.bash_profile

#=================================================================
# LAUNCH JOB
#=================================================================

echo " _ __   ___ _ __ ___   ___          "
echo "| '_ \ / _ \ '_ ' _ \ / _ \         "
echo "| | | |  __/ | | | | | (_) |        "
echo "|_| |_|\___|_| |_| |_|\___/  v3.7   "

# Go to the job submission directory and run your application
cd $PI_HOME/NEMO/nemo3.7-8666/NEMOGCM/CONFIG/GYRE_testing/EXP00/
# mpiexec here because I built bound the mpi seprately
mpiexec -n 40 ./opa

#=================================================================
# POSTPROCESSING
#=================================================================

# kills the daisy chain if there are errors

if grep -q 'E R R O R' ocean.output ; then

  echo "E R R O R found, exiting..."
  echo "  ___ _ __ _ __ ___  _ __   "
  echo " / _ \ '__| '__/ _ \| '__|  "
  echo "|  __/ |  | | | (_) | |     "
  echo " \___|_|  |_|  \___/|_|     "
  echo "check out ocean.output or stdouterr to see what the deal is "

  exit
else
```

(continues on next page)

(continued from previous page)

```
  echo "going into postprocessing stage..."
  # cleans up files, makes restarts, moves files, resubmits this pbs

  bash ./postprocess.sh >& cleanup.log
  exit
fi
```

Here because I am not using `xios_server.exe` I don't strictly need the `-n 40` after `mpirun` (it will then just use however many cores that's given in `#SBATCH -n`). Maybe see the *Oxford ARC* one to see how it might work when `xios_server.exe` is run alongside NEMO to do the I/O.

The following post-processing script requires a few prepping (I make no apologies for the bad code and the script being fickle; feel free to modify as you see fit):

- copying the `nn_date0` line into `namelist_cfg` from say `namelist_ref` if it doesn't exist already, because the time-stamps are modified by modifying `nn_date0`
- do a search in `namelist_cfg` and make sure there is only ever one mention of `nn_date0` (otherwise it grabs the wrong lines)
- `nn_date0` should not begin with zeros (e.g. `10101` rather than `010101` in `yymmdd`)
- in the experiment folder, do `mkdir RESTARTS OUTPUTS` (otherwise there is no folder to copy into)

The `postprocess.sh` I cooked up is here:

```
#!/bin/bash
#! postprocess.sh
#! Script to clean up the NEMO outputs

export BASE_DIR=$PI_HOME/NEMO/nemo3.7-8666/NEMOGCM/
export MODEL=GYRE
export NUM_CPU=40

# time-stamp increment, yymmdd
export DATE_INC=100000

# when to stop the daisy chaining, yymmdd
export THRESH=10

# error catching (only when restart files etc cannot be copied or made)
export ERR_CATCH=0

#########################################################
# 0) recombine files to one netcdf (restarts and/or outputs)
# restarts: extract the restart file time-step stamp
#              based on the *0000.nc restart which should (!) always exist
#           rebuild the restart file in the submission directory
# outputs:  put them in manually and just do a grab
#           this assumes only files at the current time-stamp is there,
#              otherwise it will bug out as it grabs wrong files
#########################################################

# restart files
export RES_TIMESTAMP=$(echo $(ls -d ${MODEL}_*_restart_0000.nc) | awk -F _ '{print $2 }')
```

(continues on next page)

(continued from previous page)

```
$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${RES_TIMESTAMP}_restart $NUM_CPU
if (($? > 0)); then
  ERR_CATCH=$((ERR_CATCH + 1))
  echo "  ERR: making the restart file in the folder"
fi
##$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${RES_TIMESTAMP}_restart_ice $NUM_
↪CPU

# output files (assumes a grid_T always exists)
export OUT_FREQ=$(echo $(ls -d ${MODEL}_*_grid_T_0000.nc) | awk -F _ '{print $2 }')
export OUT_START=$(echo $(ls -d ${MODEL}_*_grid_T_0000.nc) | awk -F _ '{print $3 }')
export OUT_END=$(echo $(ls -d ${MODEL}_*_grid_T_0000.nc) | awk -F _ '{print $4 }')

$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${OUT_FREQ}_${OUT_START}_${OUT_END}_
↪grid_T $NUM_CPU
$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${OUT_FREQ}_${OUT_START}_${OUT_END}_
↪grid_U $NUM_CPU
$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${OUT_FREQ}_${OUT_START}_${OUT_END}_
↪grid_V $NUM_CPU
$BASE_DIR/TOOLS/REBUILD_NEMO/rebuild_nemo ${MODEL}_${OUT_FREQ}_${OUT_START}_${OUT_END}_
↪grid_W $NUM_CPU

# add more things in here if output freqs are different etc

#########################################################
# 1) pull out some variables to modify namelist file
#########################################################

# pull the number out
# add the increment to it for new date
# subtract appropriately to get the date stamp
#   (e.g. 110101 - 8871 = 101230) and bulk out zeros

export OLD_DATE_STR=$(grep -ri "nn_date0" namelist_cfg)
export OLD_DATE_NUM=$(echo ${OLD_DATE_STR} | sed -e 's/[^0-9 ]//g' | awk '{print $NF}')
export NEW_DATE_NUM=$((OLD_DATE_NUM + DATE_INC))

# 8871 for 30 days a month (so the RES_STAMP=yyyy1230)
# otherwise do 8870        (so the RES_STAMP=yyyy1231)
# do something else for other time units
export RES_STAMP=$(printf %08d $((NEW_DATE_NUM - 8871)))

#########################################################
# 2) move files around and tidy up
#########################################################

cp -pv ${MODEL}_${RES_TIMESTAMP}_restart.nc ./RESTARTS/${MODEL}_${RES_STAMP}_restart.nc
cp -pv ./output.namelist.dyn ./OUTPUTS/output.namelist.dyn.${RES_STAMP}
#cp -pv ${MODEL}_${RES_TIMESTAMP}_restart_ice.nc ./RESTARTS/${MODEL}_${RES_STAMP}_
↪restart_ice.nc
#cp -pv ./output.namelist.ice ./OUTPUTS/output.namelist.ice.${RES_STAMP}
```

(continues on next page)

(continued from previous page)

```
cp -pv ./ocean.output ./OUTPUTS/ocean.output.${RES_STAMP}
cp -pv ./solver.stat ./OUTPUTS/solver.stat.${RES_STAMP}
cp -pv ./stdouterr ./OUTPUTS/stdouterr.${RES_STAMP}
cp -pv ./namelist_cfg ./OUTPUTS/namelist_cfg.${RES_STAMP}

#cp -pv ./volume_transport ./OUTPUTS/volume_transport.${RES_STAMP}
#cp -pv ./salt_transport ./OUTPUTS/salt_transport.${RES_STAMP}
#cp -pv ./heat_transport ./OUTPUTS/heat_transport.${RES_STAMP}

rm -v ${MODEL}_${RES_TIMESTAMP}_restart*
rm -v restart.nc
#rm -v restart_ice.nc
rm -v ${MODEL}_*_????.nc
mv ${MODEL}*.nc ./OUTPUTS

cp -pv RESTARTS/${MODEL}_${RES_STAMP}_restart.nc ./restart.nc
if (($? > 0)); then
  ERR_CATCH=$((ERR_CATCH + 1))
  echo "  ERR: copying restart file into folder"
fi

#cp -pv RESTARTS/${MODEL}_${RES_STAMP}_restart_ice.nc ./restart_ice.nc
#if (($? > 0)); then
#  ERR_CATCH=$((ERR_CATCH + 1))
#  echo "  ERR: copying restart_ice file into folder"
#fi

##########################################################
# 3) if all good, then modify namelist_cfg and resbumit
##########################################################

if (($ERR_CATCH > 0)) || ((${NEW_DATE_NUM} > $THRESH)); then
  if (($ERR_CATCH > 0)); then
    echo " "
    echo " "
    echo " "
    echo "ERR: caught a non-zero exit status, check cleanup.log for what the deal was"
    echo "ERR: caught a non-zero exit status, check cleanup.log for what the deal was"
  else
    echo "OK: grabbed time stamp ${NEW_DATE_NUM} larger than threshold ${THRESH},␣
↪breaking..."
    echo "OK: grabbed time stamp ${NEW_DATE_NUM} larger than threshold ${THRESH},␣
↪breaking..."
    # WARNING: this assumes that OLD_DATE_NUM is the only number within the file, which␣
↪should
    #          really be true
    sed -i "s/${OLD_DATE_NUM}/${NEW_DATE_NUM}/g" namelist_cfg
  fi
  echo " "
  echo " "
  echo " "
  echo " "
```

(continues on next page)

(continued from previous page)

```
  echo " ... a wild Totoro appeared and blocked your resubmission!"
  echo "          ,--'''',--.__,---[],-------._                                  "
  echo "        ,'    __,'             \        \--''''''==;-                   "
  echo "      ,' _,-'  '/---.___      \       ___\   ,-'','                     "
  echo "    /,-'       / ;. ,.--'-.__\  _,-'' ,| ',',   /                      "
  echo "   /''''''-._/,-|:\       []\,' '''-/:;-. '. /                           "
  echo "              '  ;:::      ||       /:,;  '-.\                          "
  echo "                =.,'__,---||-.____',.=                                   "
  echo "                =(:\_     ||__    ):)=                                   "
  echo "               ,'::::'----||::'--':::'._                                 "
  echo "             ,':::::::::::||::::::::::::'.                               "
  echo "     .__     ;:::.-.:::::__||___:::::.-.:::\     __,                     "
  echo "        '''-;:::( O ):::>_|| _<::::( O ):::-'''                         "
  echo "    =======;::::::'-':::::::||':::::::'-':::::\=======                  "
  echo "     ,--'';:::_____________||_____________:::''----.          , ,       "
  echo "          ; ::'._(    |    |||     |   )_,':::\_,,,,,,,,,,,____/,'_,   "
  echo "        ,;    :::'--._|____[]|_____|_.-'::::::::::::::::::::::::);_    "
  echo "       ;/ /      ::::::::::,||,:::::::::::::::::::::::::::::::::/      "
  echo "      /; ''''''----------/,'/,__,,,,,____:::::::::::::::::::,'         "
  echo "      ;/                :);/|_;| ,--.. . '''-.::::::::::::_,'          "
  echo "     /;                ::::):__,'//''\\. ,--.. \:::,:::::_,'            "
  echo "    ;/              :::::/ . . . . . . //''\\. \::':__,'               "
  echo "    ;/           :::::::,' . . . . . . . . . . .:'::\                  "
  echo "    ';       :::::::__,'. ,--.. . .,--. . . . . .:'::'                 "
  echo "    ';   __,..--'''-. . //''\\. .//''\\ . ,--.. :':::'                 "
  echo "    ;    /  \\ .//''\\ . . . . . . . . //''\\. :'::'                   "
  echo "    ;   /       . . . . . . . . . . . . . . . .:'::'                  "
  echo "    ;   (          . . . . . . . . . . . . . . ;:::'                  "
  echo "    ,:  ;,           . . . . . . . . . . . . . ;'::'                  "
  echo "    ,:  ;,             . . . . . . . . . . . . .;'::'                 "
  echo "    ,:   ;,             . . . . . . . . . . . . ;'::;'                "
  echo "      :   ;             . . . . . . . . . . . . ,':::;                "
  echo "       :   '.          . . . . . . . . . .. . . .,':::;'              "
  echo "        :    '.       . . . . . . . . . . . . . ;::::;'               "
  echo "         '.    '-.   . . . . . . . . . . . . ,-':::::                 "
  echo "           ':_    ''--..__________..--''::::::;''                     "
  echo "              '._::,.:,.:,:_ctr_:,:,.::,.:_;''                         "
  echo "_________________''\/'\/\/''''''\/'\/''\/'_____________________________"

else
# WARNING: this assumes that OLD_DATE_NUM is the only number within the file, which␣
↪should
  #          really be true
  sed -i "s/${OLD_DATE_NUM}/${NEW_DATE_NUM}/g" namelist_cfg

  echo "grabbed time stamp ${NEW_DATE_NUM} smaller than threshold ${THRESH},␣
↪resubmitting..."
  echo "grabbed time stamp ${NEW_DATE_NUM} smaller than threshold ${THRESH},␣
↪resubmitting..."
  echo "grabbed time stamp ${NEW_DATE_NUM} smaller than threshold ${THRESH},␣
↪resubmitting..."
  echo "grabbed time stamp ${NEW_DATE_NUM} smaller than threshold ${THRESH},␣
```

(continues on next page)

(continued from previous page)

```
↪resubmitting..."
  echo " "
  echo "OK: ...and here is Christopher resbumitting the job for you....."
  echo "                    ,-.____,-.          "
  echo "                   /   ..   \          "
  echo "                  /_        _\         "
  echo "                 |'o'      'o'|        "
  echo "                / ____________ \       "
  echo "              , ,'    '--'    '. .     "
  echo "             _| |               | |_    "
  echo "           /  ' '               ' '  \  "
  echo "          (     ',',__________.','    ) "
  echo "           \_     ' ._______, '     _/  "
  echo "              |                    |    "
  echo "              |    ,-.    ,-.     |    "
  echo "               \      ).,(      /      "
  echo "          gpyy  \___/    \___/         "
  sbatch submit_nemo

fi

exit
```

A chunk of the output recombination procedures are not required if the `one_file` option in `field_def_nemo.xml` is enabled and possible (requires parallel NetCDF4 which I didn't bother building here).

1.8 Other packages

Tested with

- `gcc4.9`, `gcc5.4` on a linux system
- `gcc4.8` on a Mac (El Capitan OSX 10.11)

The following packages are needed for NEMO and XIOS and they may need to be installed or configured accordingly. I don't have a windows machine handy (and I don't really want to try it there either) so for that I would recommend doing the following through virtualbox or something analogous (which might be another way to do it on a Mac); I am guessing *cygwin* and the new Windows 10 terminals might be a possibility.

Note: I would suggest trying the following in reverse order of effort required:

1. Get someone who knows what they are doing to do it for you! Compiling the following from scratch is not the most interesting activity and is actually quite fiddly (especially the HDF5 and NetCDF4 stuff)... if you don't have access to people who can do that, then try
2. Doing it through anaconda. There you are somewhat restricted to a certain set of compilers (`gcc 4.8`) but anaconda sorts out the dependencies for you. The only thing then you need to do is to force XIOS and NEMO to use the libraries within the anaconda installation. Failing that...
3. Do it from scratch. I'm sorry and good luck; see below for some notes to possibly ease your pain.

1.8.1 Anaconda

Anaconda is a framework mostly for downloading Python packages, with the added advantage that it resolves the package dependencies for you (cf. `apt`, `yum` on a Linux machine or `port` on a Mac if you have MacPorts). See the official conda manual or some of *my own notes* on some things to do with installing and managing conda. I used the full anaconda with Python 3.6 but you could use miniconda or with other pythons probably.

Note: [20 May 2020] Doing it through anaconda may well only work for Mac, because the gfortran versions does not seem to be available with linux through anaconda...

First I created an environment so all the changes only apply in that environment:

```
conda create -n nemo python=3.6
```

Accept to install the basic packages for the environment. Then activate the `nemo` environment with

```
>> julian@psyduck:~/
source activate nemo
>> (nemo) julian@psyduck:~/
```

Now if you have compilers you want to use already then you can skip the compiler installation. On the Mac I was dealing with there was no `gcc` or a Fortran compiler and I had problems with `clang`, so I did the following to get a set of `gcc` compilers:

```
conda install gcc
conda install gfortran_osx-64
```

The second line you should change to `gfortran_linux-64` if on a Linux machine. The command will add some compiler flags that is unset when exiting from the environment. Check that the compilers are the now default compilers by doing `gcc --version` (which should probably give 4.8) and `which gcc` (which should point to the anaconda folder). If not, do something like `echo CC` and `export CC=/folder/bin` to force it to point to the right folder (also do it for `FC` and `CXX`, and maybe put it in the `$PATH` variable; see below).

Note: One thing I found to be an issue is that while `gfortran` can compile a sample program through `gfortran hello.f90 -o hi` with `hello.f90` being

```
program hello
  print *, "hi mum"
end program hello
```

Executing through `./hi` could throw a library complaint:

```
dyld: Library not loaded: @rpath/libgfortran.3.dylib
Referenced from:
Reason: no suitable image found.  Did find:
      /usr/local/lib/libnetcdff.3.dylib: stat() failed with errno=13
```

So the problem here is that the computer is looking for the library at the wrong place. To force the computer to look at the right place, try

```
export FCFLAGS=-Wl,-rpath,${CONDA_PREFIX}/lib
```

where `${CONDA_PREFIX}` might have been defined by anaconda.

If you already have the MPI capabilities bound to the compilers you will use then you can skip the following. To make life easier it is advisable to install either MPICH or OpenMPI. You could try this by

```
conda install -c conda-forge mpich
conda install -c conda-forge openmpi
```

and check whether `which mpicc` and in particular `which mpif90`, which should be pointed to the `gcc` compilers. I had a similar problem with `gfortran` not being bound properly, which could be fixed with setting `FCFLAGS`, or to compile it from scratch (see below for the way to do it for MPICH, which also works for OpenMPI with suitable changes in the hyperlink address; do a search for this in Google).

To get NetCDF4 and its dependencies I did

```
conda install netcd4
conda install -c conda-forge netcdf-fortran
```

Do `which nc-config` and `nc-config --all` to see which paths are being pointed to. Again, you may need to add the `FCFLAGS` detailed above to make sure it is pointing to the right libraries. Take note of the path where the libraries and header files live and put those into the XIOS and NEMO files and that should be it!

1.8.2 Compiling it yourself

(Good luck!)

The following has been tried on a Linux machine. I had some problems on a Mac with `Clang` that I don't know how to fix without `sudo` access but it is probably fixable; I have not tried installing things with `port` through MacPorts partly because it requires Xcode to be installed.

A script to do all of the following on a Linux machine in one go can be found at bottom of this page. The way I went about it was to first choose a set of compilers and use the same set of compilers to install the dependencies, primarily to avoid errors relating to compatibility of packages. For example, `gcc4.9` was downloaded through `sudo apt-get install gcc4.9`, or loaded through a network computer through something like a `module load` command. You may have to look it up on the internet if you don't have either of these.

Note: If you don't have the right compilers you can always try and build your own from source, but it takes a while (order of hours) and can be quite fiddly. On e.g. HKUST HPC3 I needed some older compilers to play well with XIOS because the newer gcc compilers (version after 6) seems to be quite strict with the c++ code checking. To do this, I did

```
wget http://mirror.koddos.net/gcc/releases/gcc-5.4.0/gcc-5.4.0.tar.gz
tar -xvzf gcc-5.4.0.tar.gz
cd gcc-5.4.0/
./contrib/download_prerequisites
cd ..
mkdir gcc5.4
cd gcc5.4
../gcc-5.4.0/configure --prefix=/scratch/PI/jclmak/custom_libs/gcc5.4/ --enable-
↪languages=c,c++,fortran [--disable-multilib]
make [-j4]
make [check] install
```

The first line grabs a packaged version of gcc, in this case `5.4.0`; I chose the `x.y.0` version because I have had problems with the other versions with dependency issues with `flex` etc. (disclaimer: not checked overly rigourously because copmiling take soooo long). After unzipping, the 4th line downloads the per-requisite libraries into the source folder (gcc official website highly recommends you **do not** compile the dependencies yourselves manually).

The 6th and 7th line follows the gcc official recommendation in doing the configuring and building **not** in the source directory; change the `--prefix` to the place where you want to store the libraries, headers and binaries. The `--disable-multilib` flag forces it to build a 64-bit one only (I needed that on the particularly computer). Calling `make` will take absolutely ages (order of hours, can speed up with giving more CPUs through the `-j` flag) because it will do a bootstrap build (building needed dependencies from existing compiler then using the build tools to build the target compiler, then sorting out the dependencies with the newly built compilers); can disable but not recommended.

Once the compilers are built then proceed as usual. Of course if you are on a cluster you probably could/should get someone else to do this…

The order I did them in are:

1. mpich (to bind the set of compilers to a MPI form; I chose `mpich` but it should work on `OpenMP` too)
2. zlib (1.2.11, for HDF5)
3. hdf5 (1.8.19, for NetCDF)
4. netcdf (4.4.1.1) and netcdf-fortran (4.4.4), for XIOS

Within a folder called `gcc4.9-builds`, I added an extra `extra_variables` file containing the following:

```
export $BD=/home/julian/testing/gcc4.9-builds # CHANGE ME

export CC=/usr/bin/gcc-4.9
export CXX=/usr/bin/g++-4.9
export FC=/usr/bin/gfortran-4.9
export F77=/usr/bin/gfortran-4.9
export CPP=/usr/bin/cpp-4.9

# if you want dynamic libraries then have this
export LD_LIBRARY_PATH=$BD/install/lib:$LD_LIBRARY_PATH

# if you want static libraries then have these
export C_INCLUDE_PATH=$BD/install/include:$C_INCLUDE_PATH
export CPLUS_INCLUDE_PATH=$BD/install/include:$CPLUS_INCLUDE_PATH
export LIBRARY_PATH=$BD/install/lib:$LIBRARY_PATH

# not strictly required, only for overriding preferences in search for binary
export PATH=$BD/install/bin:$PATH
```

For my code testing it doesn't really matter too much whether the libraries are compiled as static or dynamic because I'm not hugely concerned about performance and stability, but static is probably safer. Set the above variables by doing `source extra_variables`; upon closing the terminal the variables will be flushed. Some of these may want to be added to `~/.bashrc` for convenience. The instructions below attempts to build shared rather than static libraries, and somewhat depends `LD_LIBRARY_PATH` variable being set (with the added bonus that the `ldd` command provides an extra check whether the correct libraries are being called). Suggestions on how to build the packages without setting `LD_LIBRARY_PATH` or build static packages are given below (using `LD_LIBRARY_PATH` can be dangerous, see e.g., here).

Note: Do for example `$CC --version` or `echo $CC` to see what the variables are set to. If you don't want to set the compiler variables then you need to do e.g.

```
CC=/usr/bin/gcc-4.9 FC= something ./configure something
```

where the path points to where the compiler binary lives. This then only sets the variable temporarily for the particular command.

Some or all of these may be skipped depending on which ones packages you have already installed and/or configured. The following installs all the libraries and binaries to the folder specified in `$BD`; if you have `sudo` access you install it to `/usr/local`, although I have found this can be very problematic if you need to remove the libraries (I've bricked my computer once)… The sub-directories in the folder are:

- `source`, where all the compressed files are going to live;
- `build`, where all the source file folders are going to live
- `install`, where all the compiled libraries, binaries and header files are going to live.

`source` and `build` can be deleted later.

Note: The binaries built here will not register by default unless it is added to the `$PATH` variable. If you are going to add to the `$PATH` variable, the one that gets registered **first** gets priority, i.e.

```
echo $PATH
> /home/julian/testing/gcc4.9-builds/install/bin:/usr/local/bin
```

means any binaries in `/home/julian/testing/gcc4.9-builds/install/bin` gets used first. Do this by adding to `~/.bashrc` the following:

```
export PATH=/usr/local/bin:$PATH
```

If you don't do this then it just means when you call the binaries you have to provide an explicit call, e.g., `/home/julian/testing/gcc4.9/build/bin/mpif90`. Do for example `which mpif90` to check what the `mpif90` is linked to; if you did add to `$PATH` then the `which` command above should point to the right binary.

1.8.3 MPICH

Check if there are any MPI capabilities and which compilers they are bound to:

```
mpicc --version
which mpicc
```

If you have these already they may not need to be installed. If they need to be installed separately for whatever reason, then you could do the following. I took the source files from the MPICH website itself and chose v3.0.4 here. Being in the `$BD` folder, I did:

```
cd $BD/source/
wget http://www.mpich.org/static/downloads/3.0.4/mpich-3.0.4.tar.gz
cd $BD/build/
tar -xvzf $BD/source/mpich-3.0.4.tar.gz
cd mpich-3.0.4
./configure prefix=$BD/install/
make -j 2
make check install
```

Within `install/` there should now be some folders that can be pointed to for the binaries, libraries and header files to include for later installations.

Note: The `./configure prefix=` step requires an absolute (not relative) path for the installation folder.

1.8.4 zlib and HDF5

Check whether HDF5 exists first (may still need to be installed again for compatibility reasons). `h5copy` is the command that should exist if HDF5 is installed:

```
which h5copy
h5copy --version
```

If you still want to install both zlib and HDF5, then do the following (following the instructions on the Unidata UCAR website). The raw files are taken from the HDF5 website using HDF5 v1.8.19. Again, with `$BD` as defined (don't include `-fPIC` or `--enabled-shared` if you want the libraries to be static):

```
cd $BD/source/
wget http://www.zlib.net/zlib-1.2.11.tar.gz
cd $BD/build/
tar -xvzf $BD/source/zlib-1.2.11.tar.gz
cd zlib-1.2.11
CFLAGS=-fPIC ./configure --prefix=$BD/install/
make -j 2
make check install

cd $BD/source/
wget https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.8/hdf5-1.8.19/src/hdf5-1.8.19.
↪tar.gz
cd $BD/build/
tar -xvzf $BD/source/hdf5-1.8.19.tar.gz
cd hdf5-1.8.19
#CPPFLAGS=-I$BD/install/include LDFLAGS=-L$BD/install/lib \
CFLAGS=-fPIC ./configure --enable-shared --enable-fortran --enable-cxx
--prefix=$BD/install/
make -j 2
make check install
cd $BD
```

Note: If `LD_LIBRARY_PATH` is set then zlib should be detected by the HDF5 install. If not, consider including the commented out `CPPFLAGS` and `LDFLAGS` line (the `--with-zlib` command no longer works in the newer HDF5).

HDF5 checking and installation can take a while. If it's more that 30 mins however it probably has crashed.

If a shared build option was on, then you can do `ldd h5copy` (or wherever `h5copy` is installed at if the directory has not been added to `$PATH`) to check that `libhdf5` does point to where you think it should point to. If it isn't, then try the first point in this note.

If an error shows up saying `recompile with -fPIC`, then trying doing a static build. Replace `--enable-shared` with `--disable-shared` and do the first point in this note, possibly adding `LIBS="-lz -lhdf5` etc.; see here for a guide.

1.8.5 NetCDF4

Check whether NetCDF4 exists first (may still need to be installed again for compatibility reasons). `nc-config` is the command that should exist if NetCDF4 is installed, and shows where it is installed and what compilers were used to build it.

```
nc-config all
```

If you still want to install it, then do the following (following the instructions on the Unidata UCAR website). The raw files are taken from the the NetCDF4 website, using netcdf v4.4.1.1 and netcdf-fortran v4.4.4 (don't include `-fPIC` or `--enabled-shared` if you want the libraries to be static):

```
cd $BD/source/
wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4.4.1.1.tar.gz
cd $BD/build/
tar -xvzf $BD/source/netcdf-4.4.1.1.tar.gz
cd netcdf-4.4.1.1
#CPPFLAGS=-I$BD/install/include LDFLAGS=-L$BD/install/lib \
./configure --enable-netcdf4 --enable-shared --prefix=$BD/install/
make -j 2
make check install

cd $BD/source/
wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-fortran-4.4.4.tar.gz
cd $BD/build/
tar -xvzf $BD/source/netcdf-fortran-4.4.4.tar.gz
cd netcdf-fortran-4.4.4
#CPPFLAGS=-I$BD/install/include LDFLAGS=-L$BD/install/lib \
./configure --enable-shared --prefix=$BD/install/
make -j 2
make check install
cd $BD
```

Note: NetCDF4 checking and installation can take a while. If it's more that 30 mins however it probably has crashed.

If a shared build option was on, then you can do `ldd ncdump` (or wherever `ncdump` was installed if the directory has not been added to `$PATH`) and check that `libnetcdf`, `libhdf5` and `libz` really does point to where you think it should point to. If not, consider doing something similar to the HDF5 note above.

If an error shows up saying `recompile with -fPIC`, then trying doing a static build (I had this problem on one of the computers where the Fortran part is static). See HDF5 note above.

I had a problem with not having the m4 package, which I just installed as the installation commands above, with the binaries found from `wget ftp://ftp.gnu.org/gnu/m4/m4-1.4.10.tar.gz`. This is not in the script below.

This should be it! Try `./install/bin/nc-config --all` and/or `./install/bin/nf-config --all` to see where everything is configured. The things in `build/` and `source/` may now be deleted.

1.8.6 Combined shell script

A script that does **all** of the above in one go may be found in the following commands (use at your own risk):

```
mkdir gcc4.9-builds/                # CHANGE ME
cd gcc4.9-builds/                   # CHANGE ME
wget https://raw.githubusercontent.com/julianmak/NEMO-related/master/docs/compilation_
↪notes/compile_dependencies.sh
chmod +x compile_dependencies.sh
```

Before you execute the shell script with `./compile_dependencies.sh`, make sure the compilers are pointed to appropriately. You can do this in `~/.bashrc` (see first code block on this page) or within the shell script itself (it is commented out at the moment). If some packages already exist and you don't want them installed, comment the appropriate lines.

CHAPTER

TWO

OTHER NEMO NOTES

Here you will find some of my notes relating to NEMO.

If you are interested in building a model, see how I went about doing it in the *model building* subsection.

2.1 Adding code to NEMO

2.2 Other NEMO packages

Some useful tools that come with NEMO are available in the analog of the `TOOLS` folder. These are built using the `ARCH` files as you would for building an experiment with e.g.

```
./maketools -n REBUILD_NEMO -m HKUST_HPC2
```

Notes on the ones I have used may be found here.

2.2.1 REBUILD_NEMO

XIOS can combine the output-per-CPU cells into one global file, but by default the restart files and `mesh_mask.nc` files are output per CPU, so it is useful to recombine them. This can be done through the `REBUILD_NEMO` package.

Build as usual, and the resulting output should be a `rebuild_nemo.exe` in the folder, to be driven by the script `rebuild_nemo`. The way to use it is to call the script as, for example,

```
$BASE_DIR/tools/REBUILD_NEMO/rebuild_nemo ${MODEL}_${RES_TIMESTAMP}_restart $NUM_CPU
```

where `$BASE_DIR` is wherever the folder lives, the things to be combined look like `${MODEL}_${RES_TMESTAMP}_restart_0000.nc` (e.g. `mesh_mask_00??.nc`, `UNAGI_00051840_restart[_ice].nc`, etc.), and `$NUM_CPU` are the number of files to combine (e.g. if we use 96 cores then we get `mesh_mask_0000.nc` to `mesh_mask_0095.nc`, and we should do `export NUM_CPU=96`).

Note: In NEMO 4.0 versions an error may come up with `undefined reference to iarg_` and `getarg_`. This seems to arise from `src/rebuild_nemo.f90` where both `iarg` and `getarg` are defined as `extrinsic`. With `gfortran` this seems to be fixed by simply changing the attribute to `intrinsic`.

2.2.2 SECTIONS_DIADCT

2.2.3 WEIGHTS

2.2.4 DOMAINcfg

This package generates the `domain_cfg.nc` file that encodes the grid locations and spacings, and is recommended for creating new configurations, mostly because the vertical grid spacings with partial steps correction are a bit weird to try and do manually. From the `readme` file in there, **you seem to need to use xios1 to compile this**; see the **:ref:`README <sec:nemo36:>`_** for how various things to watch out for.

When compiled the executable is called `make_domain_cfg.exe`, and it expects to read a `bathy_meter.nc` (links are ok) and a `namelist_cfg` file. The `namelist_cfg` file should contain the various settings for horizontal and vertical grid spacing, which should be consistent with the content in `bathy_meter.nc`. An example `namelist_cfg` is the following:

```
!-----------------------------------------------------------------------
&namcfg        !   parameters of the configuration
!-----------------------------------------------------------------------
   !
   ln_e3_dep   = .true.    ! =T : e3=dk[depth] in discret sens.
   !                       !      ===>>> will become the only possibility in v4.0
   !                       ! =F : e3 analytical derivative of depth function
   !                       !      only there for backward compatibility test with v3.6
   !                       !
   cp_cfg      =     "UNAGI"           !  name of the configuration
   jp_cfg      =      100              !  resolution of the configuration
   jpidta      =       90              !  1st lateral dimension ( >= jpi )
   jpjdta      =       26              !  2nd    "         "    ( >= jpj )
   jpkdta      =       31              !  number of levels      ( >= jpk )
   jpiglo      =       90              !  1st dimension of global domain --> i =jpidta
   jpjglo      =       26              !  2nd    -                   -    --> j  =jpjdta
   jpizoom     =       1               !  left bottom (i,j) indices of the zoom
   jpjzoom     =       1               !  in data domain indices
   jperio      =       1               !  lateral cond. type (between 0 and 6) [1 is EW␣
↪periodicity]
/
!-----------------------------------------------------------------------
&namzgr        !   vertical coordinate
!-----------------------------------------------------------------------
   ln_zps      = .true.    !  z-coordinate - partial steps
   ln_linssh   = .true.    !  linear free surface
/
!-----------------------------------------------------------------------
&namdom        !
!-----------------------------------------------------------------------
   jphgr_msh   =       3               !  type of horizontal mesh
   ppglam0     =       0.0             !  longitude of first raw and column T-point␣
↪(jphgr_msh = 1)
   ppgphi0     =     -50.0             !  latitude  of first raw and column T-point␣
↪(jphgr_msh = 1)
   ppe1_deg    =  999999.0             !  zonal      grid-spacing (degrees)
   ppe2_deg    =  999999.0             !  meridional grid-spacing (degrees)
```

(continues on next page)

(continued from previous page)

```
   ppe1_m      =  100000.0             !  zonal      grid-spacing (metres)
   ppe2_m      =  100000.0             !  meridional grid-spacing (metres)
   ppsur       =  999999.0             !  ORCA r4, r2 and r05 coefficients
   ppa0        =  999999.0             ! (default coefficients)
   ppa1        =  999999.0             !
   ppkth       =      18.0             !
   ppacr       =      10.0             !
   ppdzmin     =    10.0               !  Minimum vertical spacing
   pphmax      =    3000.0             !  Maximum depth
   ldbletanh   =  .FALSE.              !  Use/do not use double tanf function for␣
↪vertical coordinates
   ppa2        =  999999.0             !  Double tanh function parameters
   ppkth2      =  999999.0             !
   ppacr2      =  999999.0             !
/
```

Here, the configuration is called `UNAGI`. The `jp[ijk]data` is the number of grid cells in (x, y, z), and I chose `jp[ij]glo` to be consistent with the choice of horizontal sizes. The `jperio` denotes the periodicities (see `src/domcfg.f90` for the choices). The present model uses a Cartesian grid on a β-plane corresponding to `jphgr_msh = 3` (see `src/domhgr.f90` for choices), and is centred at longitude 0 and latitude 50 S (see `ppglam0` and `ppgphi0`). The grid spacing here is 100 km, correpsonding to `ppe[12]_m`; the values of `999999.0` are options that are not used.

For the vertical grid, `ln_zps` switches on the partial step correction and takes into account `bathy_meter.nc`. The vertical spacing is governed through the parameters `ppkth`, `ppacr`, `ppdzmin` and `pphmax` ([MI96]; unless you use the double tanh option).

Note: Note NEMO 4.2 seems to be using different namings and convention (see here). As of writing `DOMAINcfg` still reads the `jperio` option but separately defines the `l_[IJ]perio` and `ldNFold` logical flags for NEMO to read.

2.2.5 NESTING (AGRIF)

2.3 GYRE: rotated gyre model

2.3.1 Brief overview and sample outputs

2.3.2 How to get the model running

GYRE is hard-coded into NEMO so nothing needs to be provided to run it out of the box.

2.3.3 Custom analysis scripts

2.4 ORCA: global configuration

2.4.1 Brief description

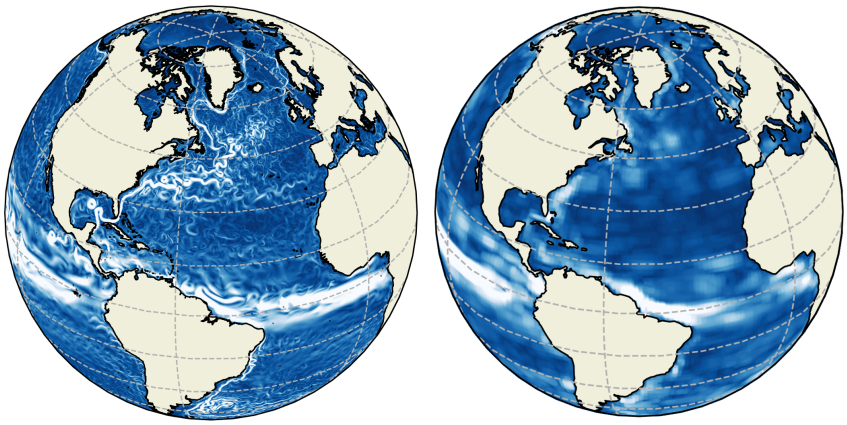


Fig. 1: Absolute speed of surface currents between ORCA at a nominal horizontal resolution of 1/12 ° and 1 ° (there is a refinement towards the equators). The ORCA 1/12 ° data was obtained from the NOC Jasmin archives.

2.4.2 How to get the model running

2.4.3 Custom analysis scripts

2.5 UNAGI: custom channel model

2.5.1 Brief overview and sample outputs

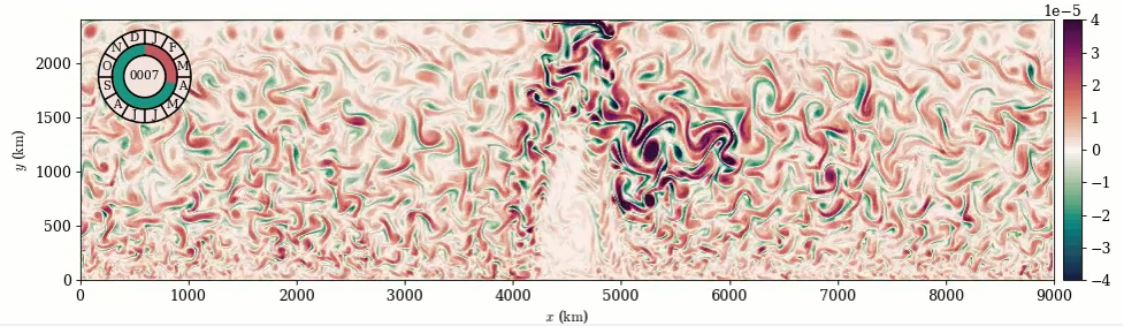
UNAGI (naming based on EEL which was original due to Marina Levy) is a re-entrant β-plane channel with temperature as the thermodynamic variable that is largely based on Dave Munday's MITgcm channel model reported in [MJM15], as an idealised model to the Antarctic Circumpolar Current. The model at present takes in some to-be specified bathymetry, wind stress profile and initial state, which may be customised accordingly within the `gen_UNAGI_fields.ipynb`, which may be found at the host GitHub repository.

With the choice of SST restoring over the surface layer, to maintain a sensible thermocline the vertical tracer diffusivity is enhanced in a sponge region to the north (see [MJM15]). See e.g. [AMF11] for alternative model formulations. Some model set up choices:

1. relatively long re-entrant zonal channel, no topography except ridge in the middle of the channel extending up to half the depth of the domain
2. fixed sinusoidal wind stress with some peak wind stress value τ_0
3. SST restoring (a relatively hard restoring, the `rn_dqdt` value in `namelist_cfg` has been amplified by a factor of 2)
4. linearly varying temperature profile at the surface with e-folding depth of 1000 metres

5. linear friction
6. linear EOS with only temperature as the thermodynamic variable
7. sponge region to the north where vertical diffusivity is amplified by a factor of 250 from the background value of 10^{-5} m^2 s^{-1}

The diagram below shows the surface relative vorticity (in units of s^{-1}) from the 10km resolution model with biharmonic tracer diffusion and no eddy parameterisation, associated with a rich eddying field. Click here for an animation.



2.5.2 How to get the model running

If you just want things to work then try the zenodo repository, which has all the NEMO modified sources files and model input files required. Some sample analysis code are given in host GitHub repository.

It is also fairly quick to recreate the forcing files from scratch, and is likely more informative for making your own models. The relevant notebook is `gen_UNAGI_fields.ipynb`, given also in host GitHub repository. The code can almost be run straight except for one step; see below notes.

2.5.3 Building the custom model

The following approach is strictly for NEMO models beyond v3.6, where one can build a customised model through providing a `domcfg.nc`, which is the main goal here. The details are given below are what I did for the idealised channel model UNAGI; see here for a step-by-step guide of how I did it.

The biggest obstacle in generating the appropriate `domcfg.nc` file for me was in transferring the code that modifies the vertical spacing variables `e3t/u/v/w` to have a partial cell description. I first tried to brute force it by writing from scratch a file that provides all the relevant variables needed in the `domcfg.nc`; see for example the input required in ORCA2. I gave up after a while and fell back to using the NEMO native [MI96] grid and the `TOOLS/DOMAINcfg` package, as follows:

1. in an external folder (e.g., `~/Python/NEMO/UNAGI`), create the bathymetry data through a program of your choice (e.g. Python), and output it as a netCDF file (e.g. `bathy_meter.nc`)
2. link/copy it as `bathy_meter.nc` (the tool requires that specific naming) into the `TOOLS/DOMAINcfg` that comes with NEMO
3. modify the `namelist_cfg` file accordingly for the horizontal and vertical grid spacing parameters (see *here* for usage and compiling notes), and the one I used for this model is given as an example in that packages page
4. a `domcfg.nc` should result (if not, see `ocean.output` for messages), copy it back into the working folder in step 1

5. open `domcfg.nc` and use those variables to create the `state.nc` and `forcing.nc` file again in the program of your choice (this is mostly to keep consistency; I did it in Python)
6. copy the `domcfg.nc`, `state.nc` and `forcing.nc` (I prefixed them with something, e.g. `UNAGI_domcfg_R010.nc`) and modify the `namelist_cfg` accordingly, e.g.

```
!-----------------------------------------------------------------------
&namrun        !   parameters of the run
!-----------------------------------------------------------------------
   cn_exp      =      "UNAGI" !  experience name
   nn_it000    =          1   !  first time step
   nn_itend    =       8640   !  last  time step
   nn_date0    =      10101   !
   nn_leapy    =         30   !  Leap year calendar (1) or not (0)
   ln_rstart   = .false.   !  start from rest (F) or from a restart file (T)
      nn_euler    =    1            !  = 0 : start with forward time step if ln_rstart=T
      nn_rstctl   =    0            !  restart control ==> activated only if ln_rstart=T
      !                             !    = 0 nndate0 read in namelist
      !                             !    = 1 nndate0 check consistancy between namelist␣
↪and restart
      !                             !    = 2 nndate0 check consistancy between namelist␣
↪and restart
   nn_stock    =       8640   !  frequency of creation of a restart file (modulo␣
↪referenced to 1)
   nn_write    =       8640   !  frequency of write in the output file   (modulo␣
↪referenced to nn_it000)
/
!-----------------------------------------------------------------------
&namcfg     !   parameters of the configuration
!-----------------------------------------------------------------------
   ln_read_cfg = .true.    !  (=T) read the domain configuration file
      !                    !  (=F) user defined configuration  ==>>>  see usrdef(_...)␣
↪modules
      cn_domcfg = "domcfg_UNAGI" ! domain configuration filename
/
...
```

That is more or less it. Once you can build the domain variables the model will at least run and the rest is more to do with experimental design.

2.5.4 Hacking NEMO to get UNAGI

That two main things that needed hacking into NEMO for UNAGI are the vertical tracer diffusion (in the sponge region to the north) and possible combination with the GEOMETRIC parameterisation, the latter could be found here. For the vertical tracer diffusion given in `zdfphy.f90`, I hacked an existing variable so that it is dual use to give a specified meridional profile in the vertical diffusivity; search for the variable `rn_avt_amp` in this file to see how I did it.

An extra hack I did was to shut off a default warning in `ldftra.f90` (with modifications to `trazdf.f90` for completeness) that biharmonic tracer diffusion cannot be used with when the GM scheme (`ldfeiv`) is used. Normally if you are using GM you also use isoneutral diffusion rather than biharmonic diffusion, but for my case I do intend on having that specific combination.

2.6 pyCDFTOOLS

For various reasons (mostly personal preference and forcing myself to write in Python) I made a translation of sorts of CDFTOOLS in Python. pyCDFTOOLS I think is:

- slightly more flexible, e.g., no need to recompile if variable name changes between files
- saves on the creation and reading of files
- everything done within Python, rather than Fortran and MATLAB say
- marginally more up-to-date, e.g. dealings with TEOS-10 equation of state

On the other hand, it is

- not as complete, because I only translated ones that I needed (see here for list)...
- not as established and probably slightly error prone
- not as fast (though things that I could not vectorise I used JIT to speed up the looping)
- not NEMO code compliant (CDFTOOLS is designed to conform to NEMO code conventions)

An additional criticism I have is that I wrote pyCDFTOOLS more like Fortran/MATLAB and not making full use of the Python functionalities (e.g., Panda and so forth). I have some idea how I might get it to work but watch this space...

The routine naming conventions of the programs are basically the same as CDFTOOLS (see MEOM page). All codes with the prefix `cdf` are based on CDFTOOLS; all errors are entirely mine (any things I did change are commented in the code).

Grab it with:

```
git clone https://github.com/julianmak/NEMO-related
```

Some slightly more configuration/model specific Python scripts and notebooks are in other folders (e.g., `GYRE` and `ORCA`). I tend to just do

```
cd GYRE
rsync -arv ../pyCDFTOOLS .
```

which then means the scripts and notebooks within the folder have access to the module, and it separates out a version that I do testing on.

CDFTOOLS itself depends on the following packages (the things I think that come as standard are omitted):

- numba (for JIT to speed up loops)
- numpy (for tools)
- netCDF4 (for reading)
- scipy (for the occasional times when a MATLAB file is read)

The configuration specific programs depend additionally on Matplotlib and a whole load of other ones for the ORCA configuration; see the relevant pages. I installed most of the things through Anaconda; see the *Python* page here for my notes on these.

Use these scripts at your own risk and feel free to modify them (rights etc. as stated in the license and in line with the CDFTOOLS one). For comparison purposes you may also want to grab CDFTOOLS to compare results (see the CDFTOOLS page):

```
git clone https://github.com/meom-group/CDFTOOLS
```

Note: The programs I have uploaded I was satisfied enough with the tests I have done, but don't just take my word for it :-)

CHAPTER

THREE

GEOMETRIC OUTLINE

TL;DR [08 Jun 2023]: My versions of the GEOMETRIC codes for NEMO and/or MITgcm can be found in this repository. The official NEMO one may be found here, with thanks to Andrew Coward at NOC-Southampton. Current version does not support the newer RK3 time-step (still needs the leap-frog), but that is on a to-do list.

GEOMETRIC (*Geometry and Energetics of Ocean Mesoscale Eddies and Their Rectified Impact on Climate*) is an approach to representing the unresolved turbulent eddies in ocean climate models, first derived in [MMB12]. David Marshall's page has an excellent outline and summary of GEOMETRIC, so this page will focus on outlining the details relating to the NEMO implementation.

The implementation of GEOMETRIC was done in NEMO by providing a new module `ldfeke.f90` and adding appropriate calls and variables to `ldftra.f90`, `step.f90` `step_oce.f90` and `nemogcm.f90`. This was initially done in SVN version 8666, which is somewhere between the 3.6 stable and 4.0 beta, by myself and Gurvan Madec back in November 2017. The current implementation of GEOMETRIC is what may be considered GM-based [GM90] and follows the prescription described in [MMMM22]. The GEOMETRIC scaling gives $\kappa_{\rm gm} = \alpha E(N/M^2)$ (see below for symbol definitions). While α is prescribed and M and N are given by the coarse resolution ocean model, information relating to E is provided by a parameterised eddy energy budget. The recipe for GEOMETRIC then is as follows:

1. time-step the parameterised eddy energy budget to get E with info provided by the GCM
2. calculate the new $\kappa_{\rm gm}$
3. use the existing GM routines with new $\kappa_{\rm gm}$ and time-step the GCM. Cycle as appropriate.

The current NEMO implementation considers an eddy energy field that varies in longitude, latitude and time (and so $\kappa_{\rm gm}$ inherits this spatio-temporal dependence), given by

$$\frac{\mathrm{d}}{\mathrm{d}t}\int E\,\mathrm{d}z + \nabla\cdot\left((\tilde{\boldsymbol{u}} - |c|\boldsymbol{e}_1)\int E\,\mathrm{d}z\right) = \int \kappa_{\rm gm}\frac{M^4}{N^2}\,\mathrm{d}z - \lambda\int E\,\mathrm{d}z + \nu_E\nabla^2\int E\,\mathrm{d}z,$$

(respecively, the time-evolution, advection, source, dissipation and diffusion of eddy energy), with $\kappa_{\rm gm}$ calculated as

$$\kappa_{\rm gm} = \alpha\frac{\int E\,\mathrm{d}z}{\int \Gamma(M^2/N)\,\mathrm{d}z}\Gamma(z).$$

The symbols are as follows:

symbol	definition	units
α	eddy efficiency parameter non-dimensional, $\lvert\alpha\rvert \leq 1$	$--$
E	total eddy energy	$m^2\ s^{-2}$
M, N	mean horizontal and vertical buoyancy gradient	s^{-1}
$\tilde{\boldsymbol{u}}$	depth-mean flow	$m^2\ s^{-1}$
$\lvert c\rvert$	magnitude of long Rossby phase speed of 1st baroclinic mode	$m^2\ s^{-1}$
$\kappa_{\rm gm}$	Gent–McWilliams coefficient	$m^2\ s^{-1}$
λ	linear damping rate of eddy energy	s^{-1}
ν_E	Laplacian diffusion of eddy energy	$m^2\ s^{-1}$

3.1 Advection

The advection of eddy energy is given in flux form and has a contribution from the depth-mean flow as well as a contribution associated with the westward propagation of eddies at the long Rossby phase speed (motivated by e.g. [CSS11] and [KM14]). The advection is by the barotropic mean flow already computed in NEMO, with a first order upwind scheme. The baroclinic Rossby wave speed is obtained by computing the eigenvalue associated with the first baroclinic mode (see e.g. eq. 6.11.8 of [Gil82]) and uses two subroutines (eke_rossby and eke_thomas) via the WKB expression given in [CdeSzoekeS+98] (their equation 2.2):

$$c_n \approx \frac{1}{n\pi}\int_{-H}^{0} N(z)\,\mathrm{d}z$$

and the long-phase speed that the total eddy energy is to be advected at is computed as (e.g. eq. 12.3.13 of [Gil82])

$$|c_p| \approx \frac{\beta}{f_0^2}c_1^2 = c_1^2\frac{\cos\phi_0}{2\Omega R\sin^2\phi_0}$$

In practice the expression diverges at the equator and the actual wave contribution to eddy energy advection as implemented in GEOMETRIC is bounded above by the magnitude tropical planetary wave phase speed (e.g. eq. 12.3.14 of [Gil82]), i.e.,

$$|c| = \min\left(|c_p|, |c_1/3|\right)$$

See here for usage and implementation details.

Note: As of Feb 2019 the removal of the routines to solve the tri-diagonal eigenvalue problem means the `nn_wav_cal` variable in `namelist_cfg` has been removed.

3.2 Source

The source of mesoscale eddy energy here is only from the slumping of neutral surfaces through the eddy induced velocity as parameterised by the GM scheme (note that it is positive-definite). These are straight-forwardly computed as is (rather than using the quasi-Stokes streamfunction) using the already limited slopes computed in NEMO. See here for implementation details.

3.3 Dissipation

The damping of eddy energy is linearly damped and the coefficient is specified in `namelist_cfg` as a time-scale in *days* (which is subsequently converted to *per seconds* in `ldf_eke_init`). There is an option to read in an externally prepared NetCDF file `geom_diss_2D.nc` that varies in longitude and latitude in anticipation of further investigation. See here for usage details, **here** for a sample Python Notebook to generate the file, and [MAD+22] and the associated Zenodo repository for some scripts to sample an estimate onto a grid onto a global grid (obtained from a finite element calculation, requires the `vtk` package in Python to probe the spherical immersed mesh).

3.4 Diffusion

The diffusion of eddy energy is through a Laplacian (cf. [EG08]), done through relevant copy and pasting of code that are in other NEMO modules. The GEOMETRIC scheme is actually stable (most likely because of the upwinding scheme). The diffusion may be switched off by setting `rn_eke_lap = 0.` in `namelist_cfg` which will bypass the relevant loop in `ldf_eke`.

CHAPTER

FOUR

MISC. CONTENT

This manual of sorts is generated using Sphinx in reStructredText, uploaded to GitHub and generated using ReadTheDocs The syntax for the relevant rst files I mostly took from the MITgcm ReadTheDocs manual. Included here **[TO DO, 04 Jul 2018]** are some notes and terminal commands I used to get the underlying python things (which acts as backend for sphinx and the sample notebooks) working.

4.1 Python / Anaconda notes

At some point I encountered some problem with plotting data in MATLAB (to do with the tripolar grid meaning the co-ordinate files were not monotonic so MATLAB hated it), and I went over to Python because the Cartopy and Iris packages lets me do data projection and plotting in different projects fairly easily. Here are some notes for Python and Anaconda which may be useful (the latter might be useful for getting the libraries that NEMO and XIOS need).

4.1.1 Anaconda

Most of these are taken from the official conda manual The installation for conda (or the lighter version miniconda) is somewhat dependent on the OS and the instructions are here You end up downloading a bash file that you run in the terminal, and from there you can accept and change some of the settings accordingly. No administrator rights should be required, though it does mean the installed packages may not be shareable. The installation will ask if you want to add to your `$PATH` variable, which I accepted (it means the some of the anaconda based binaries take precedence over the system ones).

One conda is installed, I would recommend creating an environment so that if damage is to occur, it is only within the environment which may be deleted easily without touching other things. The creation, entering and leaving of the environment is done by:

```
>> julian@psyduck:~/$ conda create -n nemo python=3.6
...
>> julian@psyduck:~/$
>> julian@psyduck:~/$ source activate nemo
>> (nemo) julian@psyduck:~/$
>> (nemo) julian@psyduck:~/$ source deactivate
>> julian@psyduck:~/$
```

The first command creates and environment called `nemo` that uses python 3.6, and the other commands are self explanatory. An environment may be removed by issuing the command

```
conda remove --name nemo --all
```

Packages are installed through (make sure you are in an environment first)

```
conda install netcdf
conda install -c conda-forge netcdf-fortran
```

Some packages need to be searched for in the forge.

Note that while the environment is active some commands take precedence over others, and a bit of care is needed to make sure the ones you intend to call really are the ones that are called (e.g. my mercurial command `hg` seems to be overwritten on my machine when I am in my environment). Check with things like `which python` for example which shows which binary the command `python` is actually calling.

4.1.2 Python

I mostly develop code in a notebook because I am too heavily influenced by MATLAB. Notebooks (in particular with Jupyter) lets you write code within cells that you run and see outputs then and there which is what I am used to. Later on I do write code in a text editor when I have more specific things I want need to do.

I normally do the following to get what I need. Within the environment:

```
conda install scipy
conda install numpy
conda install matplotlib
conda install jupyter
conda install -c conda-forge cartopy
conda install -c conda-forge iris
```

I normally install NetCDF as well. `Numpy` and `scipy` gives the number crunching stuff I normally need. `Matplotlib` gives most of the plotting capabilities. `Cartopy` and `iris` are the map and projection packages, and `jupyter` is the notebook stuff. To trigger the notebook, I normally do from a terminal

```
jupyter notebook 2>/dev/null &
```

just to suppress the terminal outputs. The notebook opens in a browser and you do coding in there (I think there is another software that lets you open and edit notebooks somewhere else though I've never used it); it's basically `ipython` but in a browser. Note that just closing the tabs does not necessarily close the notebook; you need to do `files>>close and halt`. Also, just because the relevant pages are closed in the browser does not mean the notebook server is shutdown either; you need to click `logout` on the top right corner (assuming you are not using a custom theme which suppresses that). To kill it in the terminal, either find the job through `jobs` and use `kill %n` or do

```
jupyter notebook list
>> Currently running servers:
>> http://localhost:8888/?token=7774a1ace4c2a0a1e098a5900f30c67310074a7250bd6c0d :: /
↪home/julian/GitRepo/pydra/wrapper
>> http://localhost:8889/?token=00b793728b03e2536b5a07a793bbd2a9fc1342469f3cf28d :: /
↪home/julian/Documents/NEMO

jupyter notebook stop 8888
jupyter notebook list
>> Currently running servers:
>> http://localhost:8889/?token=00b793728b03e2536b5a07a793bbd2a9fc1342469f3cf28d :: /
↪home/julian/Documents/NEMO
```

4.1.3 Some Python banana skins

The big banana skin with Python to watch out for is that indexing starts at `0` (rather than `1` in MATLAB), and index slicing normally omits the last entry, e.g.

```
x_vec = [1, 2, 3, 4, 5, 6]
x_vec[0:-1]
>> [1, 2, 3, 4, 5]
x_vec[1:4]
>> [2, 3, 4]
x_vec[0::]
>> [1, 2, 3, 4, 5, 6]
x_vec[-1]
>> 6
x_vec[-2]
>> 5
```

Contrast this to MATLAB which would be

```
x_vec = [1, 2, 3, 4, 5, 6]
x_vec(0:end-1)
>> 1, 2, 3, 4, 5
x_vec(2:4)
>> 2, 3, 4
x_vec(:)
>> 1, 2, 3, 4, 5, 6
x_vec(end)
>> 6
x_vec(end - 1)
>> 5
```

Another banana skin with python is that data is not necessarily copied when defining new variables. For example:

```
x_vec = [1, 2, 3, 4, 5, 6]
y_vec = x_vec
y_vec[0] = 2
y_vec
>> [2, 2, 3, 4, 5, 6]
x_vec
>> [2, 2, 3, 4, 5, 6]
```

This is especially dangerous if you, like me, do the following in MATLAB:

```
x_vec = zeros(6)
y_vec = x_vec
z_vec = x_vec
```

If you really mean to do a copy, do the following:

```
from copy import deepcopy
x_vec = [1, 2, 3, 4, 5, 6]
y_vec = x_vec
z_vec = deepcopy(x_vec)
y_vec[0] = 2
```

(continues on next page)

(continued from previous page)

```
y_vec
>> [2, 2, 3, 4, 5, 6]
x_vec
>> [2, 2, 3, 4, 5, 6]
z_vec
>> [1, 2, 3, 4, 5, 6]
```

Python is really slow with loops, so the more vectorising commands you can use, the better! If you have routines that you have to use loops in (e.g. transformation of data from Cartesian co-ordinates to density co-ordinates through binning into density bins), then consider using `cypthon` (write code in C but call it through Python), `f2py` (same but for Fortran), or `numba/JIT` (compile and run loops, usually on the order of 200 speed up; restricted to fairly low level commands).

4.2 sphinx notes

(working notes)

- https://github.com/ralsina/rst-cheatsheet/blob/master/rst-cheatsheet.rst
- To get bibtex working on ReadTheDocs a `requirements.txt` may be needed.

Having the following in there got the sphinx-bibtex extension working for me.

`conda install sphinx` `pip install sphinx_rtd_theme` `pip install sphinxcontrib-bibtex`

```
sphinx>=1.7.0b1
sphinxcontrib-bibtex
```

4.3 Git commands

Git is s version control software. Similar software exist (e.g. **`Mercurial <www.mercurial-scm.org >`_**, Subversion), but I almost exclusively use Git now for my own things (NEMO uses subversion but the only thing I ever do is `svn checkout LOCATION -r VERSION`, so it's not really using it...)

I personally use Git for backup mostly, occasionally reverting files, as well as hosting websites (e.g. here and here). For my kind of files (mostly text files, as source LaTeX files, html or bits of code) I find it much more convenient and safer than saying using Dropbox (manual version control is too error prone for me). I haven't personally used Git that much in terms of collabroative work at the moment, so the commands below are going to be skimpy on those related commands.

4.3.1 Repositories

Github is my go to for making repositories, partly because it can render Jupyter notebooks I use a lot. Github used to only have public repositories, but now they have private ones too so I migrated from Bitbucket. Make an account and create a repository so there is a target to push and pull files from.

Keep the files small! Github doesn't accept anything larger than 100 Mbs I think. e.g. commit LaTeX source files but not necessarily the compiled version.

4.3.2 Basic commands

The regular commands I use are:

- `git add` registers files that Git should track and note changes
- `git commit -m "SOME DEEP MESSAGE"` actually registers the changes made since last commit
- `git push [origin master]` pushes the commits up to the repository
- `git pull` pulles the commits from the repository to the local computer

Occasionally I screw something up so I need to do:

- `git mv` to move the files around by telling Git to still track them
- `git rm [--cached] FILES` to make git to stop tracking the files (the `--cached` is so that the physical files are not removed; leave it out if you actually want to get rid of it)
- `git checkout HEAD FILES` if I screw up the `rm`, `mv` or `git rm` commands to recover the removed physical files. `HEAD` can be replaced by revision number
- `git log` to check the log of commits and revision numbers

[TO ADD] some branching and merging commands

4.3.3 Access tokens

Git is phasing out password logins on terminal access, so you either have to do two factor authorisation (2FA), use a SSH key, or a token (there are others presumably). The following bits of scrap code documents how to use a token.

1. log into Git in the browswer, click your profile picture, then Settings -> Developer settings -> Personal access tokens
2. give your token a descriptive name and give permissions for the access token
3. when you click OK you should get to a screen with that tells you to copy the token (don't close this page yet!)
4. open terminal and do

```
git config --global credential.helper 'cache --timeout=31104000'
```

where you can change the `timeout` entry to something that works for you (something large if you want to keep the token active for longer, units are in seconds). I tried using `store` on Ubuntu but it doesn't seem to do anything (`store` saves an extra file with the credentials in)

5. go back to webpage, copy the access token, git as normal, but when it asks you for a password, paste the access token in instead
6. if it worked properly then now you get to bypass the username and password typing until the timeout period

At any point you can revoke the access token on the Git webpage.

BIBLIOGRAPHY

[AMF11] R. Abernathey, J. Marshall, and D. Ferreira. The dependence of Southern Ocean meridional overturning on wind stress. *J. Phys, Oceanogr.*, 41:2261–2278, 2011. doi:10.1175/JPO-D-11-023.1.

[MI96] G. Madec and M. Imbard. A global ocean mesh to overcome the North Pole singularity. *Clim. Dyn.*, 12:381–388, 1996. doi:10.1007/BF00211684.

[MJM15] D. R. Munday, H. L. Johnson, and D. P. Marshall. The role of ocean gateways in the dynamics and sensitivity to wind stress of the early Antarctic Circumpolar Current. *Paleoceanography*, 30:284–302, 2015. doi:10.1002/2014PA002675.

[CSS11] D. B. Chelton, M. G. Schlax, and R. M. Samelson. Global observations of nonlinear mesoscale eddies. *Prog. Oceanog.*, 91:167–216, 2011. doi:10.1016/j.pocean.2011.01.002.

[CdeSzoekeS+98] D. B. Chelton, R. A. de Szoeke, M. G. Schlax, K. El Naggar, and N. Siwertz. Geographical variability of the first baroclinic Rossby radius of deformation. *J. Phys. Oceanogr.*, 28:433–459, 1998. doi:10.1175/1520-0485(1998)028<0433:GVOTFB>2.0.CO;2.

[EG08] C. Eden and R. J. Greatbatch. Towards a mesoscale eddy closure. *Ocean Modell.*, 20:223–239, 2008. doi:10.1016/j.ocemod.2007.09.002.

[GM90] P. R. Gent and J. C. McWilliams. Isopycnal mixing in ocean circulation models. *J. Phys. Oceanogr.*, 20:150–155, 1990. doi:10.1175/1520-0485(1990)020<0150:IMIOCM>2.0.CO;2.

[Gil82] A. E. Gill. *Atmospheric-Ocean Dynamics*. Academic Press, 1982.

[KM14] A. Klocker and D. P. Marshall. Advection of baroclinic eddies by depth mean flow. *Geophys. Res. Lett.*, 41:L060001, 2014. doi:10.1002/2014GL060001.

[MAD+22] J. Mak, A. Avdis, T. W. David, H. S. Lee, Y. Na, and F. E. Yan. On constraining the mesoscale eddy energy dissipation time-scale. *J. Adv. Model. Earth Syst.*, 14:e2022MS003223, 2022. doi:10.1029/2022MS003223.

[MMMM22] J. Mak, D. P. Marshall, G. Madec, and J. R. Maddison. Acute sensitivity of global ocean circulation and heat content to eddy energy dissipation time-scale. *Geophys. Res. Lett.*, 49(8):e2021GL097259, 2022. doi:10.1029/2021GL097259.

[MMB12] D. P. Marshall, J. R. Maddison, and P. S. Berloff. A framework for parameterizing eddy potential vorticity fluxes. *J. Phys. Oceanogr.*, 42:539–557, 2012. doi:10.1175/JPO-D-11-048.1.