nemo-nocl Documentation

Release 0

James Harle

January 20, 2017

Contents

Contents:

CHAPTER 1

Working Environment

QuickStart Guide to working environments you may use:

1.1 NEMO on ARCHER

QuickStart Guide to setting up NEMO v3.6 and XIOS on ARCHER:

- `login.archer.ac.uk`: Access to the ARCHER HPC facility is via the login nodes. There are 8 login nodes that are automatically assigned.

Compiling the code:

At present this has been successfully acheived using the INTEL compilers (CRAY instructions to follow):

First set up the correct modules (can be added to ~/.bashrc):

```
module swap PrgEnv-cray PrgEnv-intel
module load cray-hdf5-parallel
module load cray-netcdf-hdf5parallel
```

Then obtain the code (the HEAD at time of testing was revision 6800, drop the @6800 if you want the most up-to-date version):

```
cd "my_chosen_dir"
mkdir NEMO
cd NEMO
svn co http://forge.ipsl.jussieu.fr/nemo/svn/trunk@6800
cd ../
svn co http://forge.ipsl.jussieu.fr/ioserver/svn/XIOS/branchs/xios-1.0@703
```

Next step is to compile the XIOS libraries:

```
cd xios-1.0/arch
```

Add the following code to a new file called arch-ARCHER_INTEL.path:

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

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

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

OASIS_INCDIR=""
OASIS_LIBDIR=""
OASIS_LIB=""
```

The following to arch-ARCHER_INTEL.fcm:

```
%CCOMPILER            CC
%FCOMPILER            ftn
%LINKER               ftn -nofor-main

%BASE_CFLAGS
%PROD_CFLAGS     -O3 -D BOOST_DISABLE_ASSERTS
%DEV_CFLAGS      -g -traceback
%DEBUG_CFLAGS    -DBZ_DEBUG -g -traceback -fno-inline

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

%BASE_INC        -D__NONE__
%BASE_LD         -lstdc++

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

And finally to arch-ARCHER_INTEL.env:

```
export HDF5_INC_DIR=${HDF5_DIR}/include
export HDF5_LIB_DIR=${HDF5_DIR}/lib
export NETCDF_INC_DIR=${NETCDF_DIR}/include
export NETCDF_LIB_DIR=${NETCDF_DIR}/lib
```

To compile:

```
cd ../
./make_xios --full --prod --arch ARCHER_INTEL --jobs 6
```

Now to compile the NEMO code:

```
cd ../NEMO/trunk/NEMOGCM/ARCH
```

Add a new architecture file arch-ARCHER_INTEL.fcm, being careful to replace the XIOS path entry to point to your newly compiled XIOS path:

```
# compiler options for Archer CRAY XC-30 (using intel compiler)
#
# NCDF_HOME   root directory containing lib and include subdirectories for netcdf4
# HDF5_HOME   root directory containing lib and include subdirectories for HDF5
# XIOS_HOME   root directory containing lib for XIOS
# OASIS_HOME  root directory containing lib for OASIS
#
# NCDF_INC    netcdf4 include file
# NCDF_LIB    netcdf4 library
# XIOS_INC    xios include file    (taken into accound only if key_iomput is activated)
# XIOS_LIB    xios library         (taken into accound only if key_iomput is activated)
```

```
# OASIS_INC   oasis include file   (taken into accound only if key_oasis3 is activated)
# OASIS_LIB   oasis library        (taken into accound only if key_oasis3 is activated)
#
# 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
# FPPFLAGS    pre-processing flags
# AR          assembler
# ARFLAGS     assembler flags
# MK          make
# USER_INC    complete list of include files
# USER_LIB    complete list of libraries to pass to the linker
#
# Note that:
#  - unix variables "$..." are accpeted and will be evaluated before calling fcm.
#  - fcm variables are starting with a % (and not a $)
#
%NCDF_HOME           $NETCDF_DIR
%HDF5_HOME           $HDF5_DIR
%XIOS_HOME           "path_to_xios"

%NCDF_INC            -I%NCDF_HOME/include -I%HDF5_HOME/include
%NCDF_LIB            -L%HDF5_HOME/lib -L%NCDF_HOME/lib -lnetcdff -lnetcdf -lhdf5_hl -lhdf5 -lz
%XIOS_INC            -I%XIOS_HOME/inc
%XIOS_LIB            -L%XIOS_HOME/lib -lxios

%CPP                 cpp
%FC                  ftn
%FCFLAGS             -integer-size 32 -real-size 64 -O3 -fp-model source -zero -fpp -warn all
%FFLAGS              -integer-size 32 -real-size 64 -O3 -fp-model source -zero -fpp -warn all
%LD                  CC -Wl,"--allow-multiple-definition"
%FPPFLAGS            -P -C -traditional
%LDFLAGS
%AR                  ar
%ARFLAGS             -r
%MK                  gmake
%USER_INC            %XIOS_INC %NCDF_INC
%USER_LIB            %XIOS_LIB %NCDF_LIB
```

Finally to compile the GYRE_XIOS example:

```
cd ../CONFIG
./makenemo -n XIOS_GYRE -m ARCHER_INTEL
cd GYRE_XIOS/EXP00
ln -s "path_to_xios"/bin/xios_server.exe xios_server.exe
```

Create a file called runscript.pbs in the EXP00 directory that contains:

```
#!/bin/bash --login
#PBS -l select=3
#PBS -l walltime=00:20:00
#PBS -A n01-NOCL

export OMP_NUM_THREADS=1
ulimit -s unlimited

XIOSCORES=2
```

```
OCEANCORES=9

#======================================================================
# RUN MODEL
#======================================================================

aprun -n $XIOSCORES -N 2 xios_server.exe  : -n $OCEANCORES opa

exit
```

Correct for a bug in the namelist in revision 6800:

```
sed 's/rn_ahm_0_lap/rn_ahm_0/' namelist_cfg > tmp; mv tmp namelist_cfg
```

And then submit to the compute nodes:

```
qsub -q short runscipt.pbs
```

Hopefully this is a start. More indepth instructions to follow + finer details about submitting aprun commands and the iodef.xml options!

1.2 NEMO on Mobius

QuickStart Guide to setting up NEMO v3.6 and XIOS on Mobius:

- `mobius`: Access to mobius is via:

```
ssh mobius
```

Compiling the code:

At present this has been successfully achieved using the Intel compilers:

First set up the correct modules (can be added to ~/.bashrc):

```
module purge
module load shared intel/compiler/64/14.0/2013_sp1.3.174 mvapich2/intel/64/2.0b slurm/14.03.0 cluste
```

Then obtain the code (the HEAD at time of testing was revision 6800, drop the @6800 if you want the most up-to-date version):

```
cd "my_chosen_dir"
mkdir NEMO
cd NEMO
svn co http://forge.ipsl.jussieu.fr/nemo/svn/trunk@6800
cd ../
svn co http://forge.ipsl.jussieu.fr/ioserver/svn/XIOS/branchs/xios-1.0@703
```

Next step is to compile the XIOS libraries:

```
cd xios-1.0/arch
```

Add the following code to a new file called arch-mobius_intel.path:

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

MPI_INCDIR=""
```

```
MPI_LIBDIR=""
MPI_LIB=""

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

OASIS_INCDIR=""
OASIS_LIBDIR=""
OASIS_LIB=""
```

The following to arch-mobius_intel.fcm:

```
%CCOMPILER            mpicc
%FCOMPILER            mpif90
%LINKER               mpif90 -nofor-main

%BASE_CFLAGS
%PROD_CFLAGS     -O3 -D BOOST_DISABLE_ASSERTS
%DEV_CFLAGS      -g -traceback
%DEBUG_CFLAGS    -DBZ_DEBUG -g -traceback -fno-inline

%BASE_FFLAGS     -D__NONE__
%PROD_FFLAGS     -O3
%DEV_FFLAGS      -g -O2 -traceback -fp-stack-check -check bounds
%DEBUG_FFLAGS    -g -traceback

%BASE_INC        -D__NONE__
%BASE_LD         -lstdc++

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

And finally to arch-mobius_intel.env:

```
export HDF5_INC_DIR=/login/jdha/utils/hdf5_mob_intel/include
export HDF5_LIB_DIR=/login/jdha/utils/hdf5_mob_intel/lib
export NETCDF_INC_DIR=/login/jdha/utils/netcdf_mob_intel/include
export NETCDF_LIB_DIR=/login/jdha/utils/netcdf_mob_intel/lib
```

To compile:

```
cd ../
./make_xios --full --prod --arch mobius_intel --jobs 6
```

Now to compile the NEMO code:

```
cd ../NEMO/trunk/NEMOGCM/ARCH
```

Add a new architecture file arch-mobius_intel.fcm, being careful to replace the %XIOS path entry to point to your newly compiled XIOS path:

```
# compiler options for Mobius (using intel compiler)
#
# NCDF_HOME   root directory containing lib and include subdirectories for netcdf4
# HDF5_HOME   root directory containing lib and include subdirectories for HDF5
# XIOS_HOME   root directory containing lib for XIOS
# OASIS_HOME  root directory containing lib for OASIS
#
```

```
# NCDF_INC    netcdf4 include file
# NCDF_LIB    netcdf4 library
# XIOS_INC    xios include file    (taken into accound only if key_iomput is activated)
# XIOS_LIB    xios library         (taken into accound only if key_iomput is activated)
# OASIS_INC   oasis include file   (taken into accound only if key_oasis3 is activated)
# OASIS_LIB   oasis library        (taken into accound only if key_oasis3 is activated)
#
# 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
# FPPFLAGS    pre-processing flags
# AR          assembler
# ARFLAGS     assembler flags
# MK          make
# USER_INC    complete list of include files
# USER_LIB    complete list of libraries to pass to the linker
# CC          C compiler used to compile conv for AGRIF
# CFLAGS      compiler flags used with CC
#
# Note that:
#  - unix variables "$..." are accpeted and will be evaluated before calling fcm.
#  - fcm variables are starting with a % (and not a $)
#
%NCDF_HOME           /login/jdha/utils/netcdf_mob_intel
%HDF5_HOME           /login/jdha/utils/hdf5_mob_intel
%XIOS_HOME           "path_to_xios"

%NCDF_INC            -I%NCDF_HOME/include -I%HDF5_HOME/include
%NCDF_LIB            -L%HDF5_HOME/lib -L%NCDF_HOME/lib -lnetcdff -lnetcdf -lhdf5_hl -lhdf5 -lz
%XIOS_INC            -I%XIOS_HOME/inc
%XIOS_LIB            -L%XIOS_HOME/lib -lxios

%CPP                 cpp
%FC                  mpif90 -c -cpp
%FCFLAGS             -i4 -r8 -O3 -fp-model source -fpp -warn all
%FFFLAGS             -i4 -r8 -O3 -fp-model source -fpp -warn all
%FPPFLAGS            -P -C -traditional
%LD                  mpif90
%LDFLAGS             -lstdc++
%AR                  ar
%ARFLAGS             -r
%MK                  gmake
%USER_INC            %XIOS_INC %NCDF_INC
%USER_LIB            %XIOS_LIB %NCDF_LIB
```

Finally to compile the GYRE_XIOS example:

```
cd ../CONFIG
./makenemo -n GYRE_XIOS -m mobius_intel -j 10
cd GYRE_XIOS/EXP00
ln -s "path_to_xios"/bin/xios_server.exe xios_server.exe
```

Create a runscript in the EXP00 directory that contains:

```
#!/bin/bash
# ---------------------------
#=============================================================
```

```
# CLUSTER BITS
#=============================================================
#
#SBATCH -J NEMO
#SBATCH -p inter
#SBATCH -N 3
#SBATCH -n 11

module purge
module load shared intel/compiler/64/14.0/2013_sp1.3.174 mvapich2/intel/64/2.0b slurm/14.03.0 cluste

XIOSCORES=2
OCEANCORES=9

#=============================================================
# GATHER NODE INFORMATION
#=============================================================

hostlist=$(scontrol show hostname $SLURM_JOB_NODELIST)
rm -f hostfile
l=0
for f in $hostlist
do
  if [ $l -lt $XIOSCORES ]; then
    echo $f':1' >> hostfile
  else
    echo $f':16' >> hostfile
fi
let l++
done
arr=($hostlist)
echo ${arr[@]:0:$XIOSCORES}

#=============================================================
# RUN MODEL
#=============================================================
# Change to the directory that the job was submitted from

cd $SLURM_SUBMIT_DIR
echo 'In derectory: ' $SLURM_SUBMIT_DIR
export OMP_NUM_THREADS=1

time mpirun -f hostfile -np $XIOSCORES xios_server.exe  : -np $OCEANCORES opa
# The use of the hostfile is to distribute the XIOS servers evenly
# if not required try the following instead:
#
# time mpirun -np $XIOSCORES xios_server.exe  : -np $OCEANCORES opa
exit
```

Correct for a bug in the namelist in revision 6800:

```
sed 's/rn_ahm_0_lap/rn_ahm_0/' namelist_cfg > tmp; mv tmp namelist_cfg
```

And then submit to the compute nodes:

```
sbatch runscipt.pbs
```

Hopefully this is a start. More indepth instructions to follow + finer details about submitting aprun commands and the iodef.xml options!

1.3 bashrc

1.4 SSHFS & FUSE

"This is a filesystem client based on the SSH File Transfer Protocol. Since most SSH servers already support this protocol it is very easy to set up: i.e. on the server side there's nothing to do"

Why use sshfs at NOCL?

If you need access to large data sets on remote file systems, using sshfs will allow do this without having to download the data locally.

How do I use sshfs at NOCL?

- First you will need a user id on the remote file system.
- Password-less login may also been helpful here `man ssh-keygen`
- Check your system has fuse and sshfs available `which fusermount sshfs`
- Create an empty directory to provide the mount point for the remote filesystem.
- Issue the command `sshfs path_to_remote_dir path_to_mount_point -o default_permissions,uid=XXXX,gid=YYYY,umask=022'`. XXXX and YYYY are your user id and group id (see below).
- To unmount the remote filesystem issue the command `fusermount -u path_to_mount_point`

A simple example to illustrate:

In my `.bashrc` I've set up an alias to mount and an alias to unmount the remote hector filesystem.

```
alias mnt_rmt='sshfs jdha@supercomputer.ac.uk:/work /work/jdha/mount_points/supercomuter -o default_p
alias umnt_rmt='fusermount -u /work/jdha/mount_points/supercomputer'
```

In mounting the filesystem you have to pass your user id (`uid`) and group id (`gid`). If you type `id` on the command line you will be able to identify your `uid` and `gid`. You will then have to request that your `uid` is added to the fusergroup (contact NOCL help desk).

The directory `/work/jdha/mount_points/supercomputer` is an empty directory that is created for the sole purpose of mounting the remote filesystem and should not be populated.

Points to note:

1. Do not locate your mount points on `/login` - if the remote filesystem is large. Any requests to the remote filesystem, even a basic ls, will grind things to a halt. This happens as `/login` is not part of GPFS and cannot handle the overhead. I would suggest providing empty mount points on `/work`.
2. You can use ssh passwordless login by generating ssh keys for local and remote sites to get around being prompted for your password each time you want to mount the remote filesystem.
3. If the filesystem you are trying to access happens to be behind a *login* machine you will first have to tunnel through to provide access:

- first set up the tunnel: `ssh -f login.machine.ac.uk -L 2221:remote_machine_name:22 -N`
- then mount as before: `sshfs -p 2221 -o transform_symlinks localhost:/work/jdha/mount_points/my_mount_point/`. The choice of port number is arbitrary.

CHAPTER 2

Configurations

List of Configs

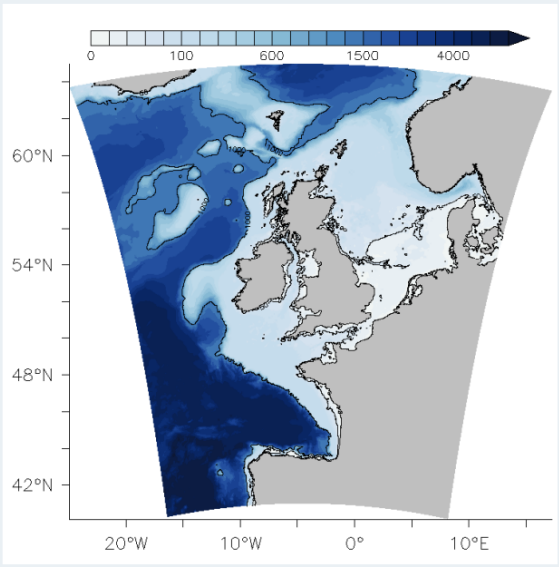
2.1 Atlantic Margin Model 7km (AMM7)

Summary + pic

2.1.1 Domain

2.1.2 References

2.2 Atlantic Margin Model 1/60 (AMM60)



Bathymetry of the domain (metres).

Domain:
- ≈ same domain as AMM7
- Regular grid ≈ 1.8 km
- bathymetry from GEBCO

Parameterisation:
- NEMO 3.6
- 2000 processors
- 51 hybrid-σ-levels

Forcings (2010):
- Surface forcings: ERA-interim (CORE bulk formulation)
- Tidal forcing: TPXO7.2
- BDY forcings: ORCA0083

2.2.1 Domain

The AMM60 configuration is based on the *Atlantic Margin Model 7km (AMM7)* domain, which spans the region from 340°W-40°N to 13°E- 65°N. The NEMO tripolar grid is rotated such that the equator passes through the domain. This operation results in grid with a resolution of 1.8 km in both pseudo north and east directions.

2.2.2 References

Warner, J.C., Sherwood, C.R., Arango, H.G., and Signell, R.P.: Performance of four turbulence closure models implemented using a generic length scale method, Ocean Modelling, 8, 81-115, 2005.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search