NCAR

UCAR

Porting CIME based models

Jim Edwards
CESM Software Engineering Group

NCAR is sponsored by National Science Foundation

CGD Climate & Global Dynamics a Laboratory of NCAR

7/14/2023

Options for porting CIME

1. Using the predefined ‘homebrew’ or ‘centos7-linux’ definitions
   - A good option for personal use on a mac or centos based system with gnu compilers
   - May require root access to the system to install prerequisites
2. Defining a new machine locally in your $HOME/.cime directory
   - A good option for personal use when option 1 won’t work.
3. Defining a new machine and submitting a CIME pull request
   - Use this when the system is to be shared among multiple users.

The $HOME/.cime directory

When you use a CIME model it will look for a directory $HOME/.cime

You may put several files in that directory for CIME to use.

1. config: custom settings used by the python scripts
2. config_machines.xml: custom machine description
3. config_compilers.xml: custom compiler instructions
4. config_batch.xml: custom batch system instructions

Using homebrew or centos7-linux

1. homebrew (for macs)
   - Install homebrew https://brew.sh
   - Install gcc
   - Install netcdf and mpich with the --build-from-source -cc=gcc-8 options
   - Create directories $HOME/projects/scratch $HOME/projects/cesm-inputdata
   - Use the --machine homebrew option to create_newcase
2. centos7-linux
   - Install modules (modules.sourceforge.net)
   - Install gcc, mpich, netcdf as modules
   - Create directories $HOME/cesm/inputdata $HOME/cesm/scratch
   - Use the --machine centos7-linux option to create_newcase

Steps for Porting CIME using files in $HOME/.cime

1. Copy cime/config/xml_schemas/config_machines_template.xml to $HOME/.cime/config_machines.xml
2. Edit that file and change values to fit your machine
3. Create, build and submit a case.
   a. If build errors occur due to incompatible compiler flags edit the file Macros.make to resolve.
   b. If submit errors occur due to incompatible batch system flags edit env_batch.xml to resolve.
4. Once you can successfully build and submit a case transfer the changes in Macros.make to $HOME/.cime/config_compilers.xml and those in env_batch.xml to $HOME/.cime/config_batch.xml
5. Run CIME system tests to verify port.

config

The config file may contain custom settings for several variables used by the case control (python) code, it may also be used to customize the case control code logging system.

Most ports will not require this file.

config_machines.xml

This file contains most of the details CIME needs to know about your machine.

If your machine is not defined by the default config_machines.xml file this is where you should begin.

You may start by copying the file
cime/config/xml_schemas/config_machines_template.xml

to $HOME/.cime/config_machines.xml

You will then edit the file and fill in the required information.

config_machines.xml

These file header lines should not require any changes:

```
<?xml version="1.0"?>
<!-- This is an ordered list, not all fields are required, optional fields are noted below. -->
<config_machines version="2.0">
<!-- MACH is the name that you will use in machine options -->
```

The machine definition starts with a name and a description, the name must be unique in config_machines.xml:

```
<machine MACH="mymachine">
  <!-- DESC: a text description of the machine, this field is current not used in code-->
  <DESC>SITE VENDOR platform, os is ---, xx pes/node, batch system is ---</DESC>
```

The next field is an optional identifier, NODENAME_REGEX which will make the --machine command line entry optional

```
<!-- NODENAME_REGEX: a regular expression used to identify this machine
  it must work on compute nodes as well as login nodes, use machine option
  to create_test or create_newcase if this flag is not available -->
  <NODENAME_REGEX>.*.cheyenne.ucar.edu</NODENAME_REGEX>
```

config_machines.xml

Next we need to indicate what Operating System is used here:

```
<!-- OS: the operating system of this machine. Passed to cppflags for
        compiled programs as -DVALUE  recognized are LINUX, AIX, Darwin, CNL -->
<OS>LINUX</OS>
   <!-- PROXY: optional http proxy for access to the internet-->
<PROXY> https://howto.get.out </PROXY>
```

COMPILERS is a list of compiler names available on this system, these must each match a compiler name in config_compilers.xml

```
<!-- COMPILERS: compilers supported on this machine, comma separated list, first is default -->
<COMPILERS>intel,gnu</COMPILERS>
```

MPILIBS

```
<!-- MPILIBS: mpilibs supported on this machine, comma seperated list,
     first is default, mpi-serial is assumed and not required in this list-->
<MPILIBS>mpt,openmpi,impi</MPILIBS>
```

PROJECT (optional)

```
<!-- PROJECT: A project or account number used for batch jobs
        can be overridden in environment or $HOME/.cime/config -->
<PROJECT>couldbethis</PROJECT>
```

config_machines.xml

```
<!-- CIME_OUTPUT_ROOT: Base directory for case output,
        the case/bld and case/run directories are written below here -->
   <CIME_OUTPUT_ROOT>/glade/scratch/$USER</CIME_OUTPUT_ROOT>

   <!-- DIN_LOC_ROOT: location of the inputdata data directory
        inputdata is downloaded automatically on a case by case basis as
        long as the user has write access to this directory.   We recommend that
        all cime model users on a system share an inputdata directory
        as it can be quite large -->
   <DIN_LOC_ROOT>$ENV{CESMDATAROOT}/inputdata</DIN_LOC_ROOT>

   <!-- DOUT_S_ROOT: root directory of short term archive files, short term
     archiving moves model output data out of the run directory, but
     keeps it on disk-->
   <DOUT_S_ROOT>$CIME_OUTPUT_ROOT/archive/$CASE</DOUT_S_ROOT>
   <!-- BASELINE_ROOT:  Root directory for system test baseline files -->
   <BASELINE_ROOT>$ENV{CESMDATAROOT}/cesm_baselines</BASELINE_ROOT>
   <!-- CCSM_CPRNC: location of the cprnc tool, compares model output in testing-->
   <CCSM_CPRNC>$ENV{CESMDATAROOT}/tools/cime/tools/cprnc/cprnc.cheyenne</CCSM_CPRNC>
```

config_machines.xml

```
<<!-- GMAKE: gnu compatible make tool, default is 'gmake' -->
    <GMAKE></GMAKE>
  <!-- GMAKE_J: optional number of threads to pass to the gmake flag -->
    <GMAKE_J>8</GMAKE_J>
```

BATCH_SYSTEM should match a name in config_batch.xml

```
    <!-- BATCH_SYSTEM: batch system used on this machine,
    supported values are: none, cobalt, lsf, pbs, slurm -->
   <BATCH_SYSTEM>pbs</BATCH_SYSTEM>

  <!-- SUPPORTED_BY: contact information for support for this system
    this field is not used in code -->
   <SUPPORTED_BY>cseg</SUPPORTED_BY>
```

config_machines.xml

```
<!-- MAX_TASKS_PER_NODE: maximum number of threads*tasks per
       shared memory node on this machine,
       should always be >= PES_PER_NODE -->
  <MAX_TASKS_PER_NODE>36</MAX_TASKS_PER_NODE>

  <!-- PES_PER_NODE: number of physical PES per shared node on
       this machine, in practice the MPI tasks per node will not exceed this value -->
  <MAX_MPITASKS_PER_NODE>36</MAX_MPITASKS_PER_NODE>
```

```
  <!-- PROJECT_REQUIRED: Does this machine require a project to be specified to
       the batch system?  See PROJECT above -->
  <PROJECT_REQUIRED>TRUE</PROJECT_REQUIRED>
```

config_machines.xml

```
<!-- mpirun: The mpi exec to start a job on this machine, supported values
            are values listed in MPILIBS above, default and mpi-serial -->
        <mpirun mpilib="mpt">
          <!-- name of the exectuable used to launch mpi jobs -->
          <executable>mpiexec_mpt</executable>
          <!-- arguments to the mpiexec command, the name attribute here is ignored-->
          <arguments>
                  <arg name="ntasks">-np {{ total_tasks }} </arg>
             <arg name="labelstdout">-p "%g:"</arg>
             <arg name="threadplacement"> omplace </arg>
          </arguments>
        </mpirun>
```

config_machines.xml

Software module systems

```
<!-- allowed types are none, module, soft -->
 <module_system type="module">
 <modules>
   <command name="purge"/>
   <command name="load">ncarenv/1.0</command>
</modules>
<modules compiler="intel">
   <command name="load">intel/16.0.3</command>
   <command name="load">mkl</command>
</modules>
  </module system>
```

config_machines.xml

```
<!-- environment variables, a blank entry will unset a variable -->
        <environment_variables>
          <env name="OMP_STACKSIZE">256M</env>
          <env name="MPI_TYPE_DEPTH">16</env>
        </environment_variables>
```

Closure:

```
    </machine>
</config_machines>
```

Test and confirm xml correctness:

```
xmllint --noout --schema cime/config/xml_schemas/config_machines.xsd
$HOME/.cime/config_machines.xml
```

Expected result:

```
$HOME/.cime/config_machines.xml validates
```

config_compilers.xml

The config_compilers.xml file has a default definition for each supported compiler.

This definition can be amended with a machine specific section:

```
<compiler MACH="hobart" COMPILER="nag">
  <CPPDEFS>
    <!-- needed for nag pio build.. -->
    <append> -DNO_C_SIZEOF </append>
  </CPPDEFS>
  <LDFLAGS>
    <append> -lpthread</append>
  </LDFLAGS>
  <SLIBS>
    <append> -L/usr/local/nag/lib/NAG_Fortran </append>
  </SLIBS>
</compiler>
```

config_compilers.xml

Test and confirm xml correctness:

xmllint --noout --schema cime/config/xml_schemas/config_compilers.xsd
$HOME/.cime/config_compilers.xml

Expected result:
$HOME/.cime/config_compilers.xml validates

In addition make sure that the generated Macros.make is as expected.

config_batch.xml

Default definitions for each batch queueing system which can be amended.

```
<!-- bluewaters is PBS -->
 <batch_system MACH="bluewaters" type="pbs" >
   <directives>
     <directive>-l nodes={{ num_nodes }}:ppn={{ tasks_per_node }}:xe</directive>
     <directive default="/bin/bash" > -S {{ shell }} </directive>
   </directives>
   <queues>
     <queue walltimemax="24:00:00" default="true">regular</queue>
     <queue walltimemax="00:30:00" jobmin="1" jobmax="512">debug</queue>
   </queues>
 </batch_system>
```

scripts_regression_tests.py

- A porting test for basic model functionality.
- requires that you build and install cprnc
- optionally uses pylint (install using pip or conda)
- Run from cime/scripts/tests

The CESM UltraFast Ensemble Consistency Test

With this test you will conduct three model runs of 9 timesteps each.

Postprocess the output files to add metadata about the origin of the run.

Upload to http://www.cesm.ucar.edu/models/cesm2/verification/

Your runs are then compared to an ensemble of about 300 members.

Pass or Fail is determined by whether your runs fit within the bounds of the ensemble.

See http://www.cesm.ucar.edu/models/cesm2/python-tools/ for details

Resources for help

CGD Forum: http://forum.cgd.ucar.edu/

ESCOMP development (CESM): https://github.com/ESCOMP/CESM

ESMCI development (CIME): https://github.com/ESMCI/cime

Changes in CESM2.3+

- MCT coupler/driver has been replaced with ESMF based CMEPS driver and mediator
- New CDEPS data model components
- config_compilers.xml replaced with cmake modules
- More interchangeable component options
- Much easier to introduce new grids
  - offline land fraction and mapping files are no longer required - this reduces the number of required grid files from ~25 to 4
-

CESM on your Laptop

With containers we can provide the community with *ready-to-run* CESM software - porting is much simpler!

They're also *portable* across Mac, Windows and Linux.

One popular variant ("CESM-Lab") includes a Jupyter Lab environment and a basic tutorial.

Great for laptops / desktops, and thus simple or low-res models. HPC version for clusters is being developed.

CESM on the Cloud

If you don't have a local cluster, CESM can also run on the cloud - on AWS now, and Azure soon.

Also fully preconfigured; no porting needed.

However, cloud *costs* are considerably higher than academic HPC systems, which limits use cases.

Public gateway / tool coming later this year.

Questions / Comments?

jedwards@ucar.edu

NCAR is sponsored by National Science Foundation