Porting and Enabling Use of the Community Earth System Model on ARCHER

Gavin J. Pringle, EPCC

Version 1.0, 24th September, 2015

1 Introduction

The Community Earth System Model (CESM) is a coupled climate model for simulating the earth's climate system. Composed of four separate sub-models simultaneously simulating the earth's atmosphere, ocean, land surface ice and sea ice, and one central coupler component, CESM allows researchers to conduct fundamental research into the earth's past, present and future climate states [1].

This report describes the work undertaken under the embedded CSE programme of the ARCHER UK National Supercomputing Service [2], and was entitled “Porting and Enabling use of the Community Earth System Model on ARCHER”, where the PIs were Dr Massimo Bollasina and Dr Mike Mineter, University of Edinburgh, with the technical work undertaken by the author, between the beginning of April and mid-November, 2014.

Two versions have been ported to ARCHER for all ARCHER users, namely version 1.2.2, the latest version, and version 1.0.6, a popular yet older version.

1.1 CESM

The Community Earth System Model (CESM) is one of the leading global climate models, widely recognized and successfully used by the international research community.

The method of installing CESM is different to most other software packages on ARCHER [2], a Cray XC30, in that users have access to the source code and build the specific executables depending on the target simulation. As such, CESM is not available via modules; however, there is an exhaustive, highly detailed User Guides made available for any ARCHER user to install CESM v1.0.6, [3], and v1.2.2, [4].

1.2 Technical Information

CESM already runs on a wide range of supercomputers (see [ref5] for CESM 1.2.0), including another Cray XC30, namely Edison in the USA, (see [ref6] for a report on CESM atmosphere model), and on a Cray XE6 system, namely HECToR in the UK [ref7]. For this reason, and given the evident active cooperation across the CESM users community, the risks associated with porting seemed low; and testing and profiling were expected to dominate the use of time in this project.

CESM comprises about one million lines of code (mainly Fortran90), and about twelve thousand lines of scripts. It also comprises five different sub-models (land, ocean, atmosphere, land ice and sea ice), each needing pools of parallel processes, of potentially different sizes.

The model components periodically stop to exchange information with the coupler. The coupler receives fields from the component models, computes, maps, and merges this information, then sends the fields back to the component models. The coupler brokers this

sequence of communication interchanges and manages the overall time progression.

1.3 Work plan

Initially, the aim was to make a single installation (source code and executable(s)) available to the National Centre of Atmospheric Science ARCHER group only; however this was altered in the following manner. We have not made source or executables available to others as a complete package via modules, say, because each user will have distinct needs. Further, we generated code modifications or setup scripts that run on ARCHER and, through the ARCHER website, all other ARCHER groups can get access to these. The main issue here was that we could not treat CESM as one black box: people add/subtract code and run it differently enough that we could not have a reference source copy on ARCHER

The original work plan was to employ CESM versions 1.0.5 and 1.2.0: the former because of the legacy of CMIP5 runs, and the latter bringing in new key features which are currently being validated by the CESM community.

As part of the testing procedure, we employed two large cases of direct interest to the PIs: an Atmosphere-only case and a recent Coupled case as employed in the Intercomparison Project (CMIP5), part of the latest (September 2013) IPCC fifth assessment report. These two large cases will be referred to as Case 1 and Case 2, respectively, from hereon.

Case 1 is the simplest, with relatively fast run time, comprising only the atmosphere and land models.

The Case 1 has a resolution of f19_f19 and a *compset* of F_AMIP_CAM5, where resolution of the global mesh and *compset* is a CESM flat which determines which modelling components are enabled.

Case 2 is the most complex, using all CESM component sub-models with the addition of atmospheric chemistry to represent aerosol processes in the atmosphere. Successfully running the second configuration means that a wide range of other configurations can be effectively installed on ARCHER.

The Case 2 case has a resolution f19_g16 and a compset of B_1850_CAM5_CN.

Changes to the Initial Work Plan

During the initial stages of the project, it was decided to change the original work plan in a number of ways.

Firstly, during the initial phase of the project, after 1.0.5 was compiled but untested on ARCHER, more reliable forms of the two versions were released. As such, the project switched versions: v1.0.6 for v1.0.5 and v1.2.2 for v1.2.0. (Version 1.0.6 contained new physics and Edison-specific files, and this was of interest since both Edison and ARCHER are both Cray XC30 platforms.)

Secondly, during the installations of both v1.0.6 and v1.2.2, it was found that the given default version of the code contained compiler flags used for debugging, namely –g and –ftz, where the former enables source-code debugging, and the latter checks every floating point operation. Employing these flags tends to slow execution. As such, new tasks were introduced to attempt to optimise the default installation in two methods: replacing debugging compiler flags with those for optimization; and, furthermore, replacing the netcdf library with the parallel-netcdf library, to provide a parallel I/O strategy.

In the initial Work Plan, Case 2 had a finer grid; however, we choose a relatively coarse spatial resolution (about 2°x2°, equivalent to approximately 200x200 km in the horizontal for the atmosphere) in order to reduce run-times during testing.

Lastly, it was found that CESM comes with a Port Validation process [ref5], which provides are far more detailed and robust method than the methodology in this project's original Work Plan to ensure the porting process has been successful. Thus, the work plan was expanded to include this official Port Validation process.

CESM Port Validation process

Using CESM's Port Validation process proved to be problematic.

For instance, the HTML version of the User Guide linked to the Port Validation process clearly; however, the hot link to this process was not visible in the PDF version of the User Guide and can be missed by users employing the PDF version only. Moreover, the process was found to contain bugs and typographical errors, which suggested to

the author that the process is perhaps not as widely employed as first thought. These errors were reported to the CESM team.

Using the new test suite, the author found that the "short-term archive" must be created before the configure/build/test process is started, so CESM has to be reinstalled from scratch.

We also found that, for v1.0.6, the third task involves four test cases; however the instructions for v1.0.6 are located in v1.0.4 User Guide, and v1.0.4 employs CAM3.0 whilst v1.0.6 employs CAM5.0; thus the links in the User Documentation to trusted output, employed for comparison, were not relevant.

The CESM Validation Procedure, as described in the CESM User Guide, [8], is far more extensive and thorough than any other HPC simulation packages seen by either the author nor the PIs, and would consume far more cycles than envisioned at first glance. As such, the Validation Procedure outlined in our Revised Work plan was a reduced form of the actual CESM Validation Procedure in order to reduce the amount of cycles required by this project, whilst retaining the same confidence that the port is valid.

The official CESM Port Validation has 5 Tests, where Test 1 contains 11 functionality cases, Test 2 performs scaling tests, Test 3 contains both functional and scientific validation tests, Test 4 performs two one-year runs and finally Test 5 performs a 20-30 year run.

Our revised testing procedure employing 4 Tasks, and is as follows:

- Task 1: as per the original Test 1, wherein the eleven cases from Test 1 are investigated in full.
- Task 2: as per the original Test 2, wherein scaling tests are undertaken, but with an extended set of tests, namely Cases 7 and 8 from Test 1 and our own large scale models, namely Case 1: the Atmosphere-only model, and Case 2: and the Coupled model. The two large-scale cases are run for 5 days of simulated time.
- Task 3: as per Test 5, however, we replace the given 20 year runs with our two target models, namely Case 1 and Case 2, both simulating 20 years.
- Task 4: as per Test 3, but only if Task 3 fails.

2 Experiences when installing CESM

2.1 Version 1.0.6

The default version of CESM v1.0.6 has now been ported to ARCHER, using the following modules:

```
export CRAYPE_LINK_TYPE=dynamic
module load cmake
module load svn
module swap PrgEnv-cray PrgEnv-intel
module load cray-netcdf/4.3.2
module load cray-parallel-netcdf/1.4.1
module load cray-hdf5/1.8.13
```

Version 1.0.6 installed on ARCHER, using serial I/O, using Edison 1.0.6 and HECToR 1.0.5 files as guides. Where possible, the method employed for HECToR is retained to provide users with a similar installation to HECToR to permit a smooth transition.

A number of issues arose when porting the riven default v1.0.6 to ARCHER.

It was found that the *cprnc* tool must be built by hand *before* CESM is installed, and its location appended the executable name to its PATH in the config_machines.xml file. The user guide states that the PATH should be updated, and doesn't mention having to build cprnc by hand before installation.

Further, for Port Validation, Test 1, the cases 3, 7, 9 and 11 all need the high-memory nodes (whilst the others can run on the normal-sized memory nodes). Cases 6, 7 and 11, again in Test 1, needed a further debug flag, "-traceback", added to the F90 compilation flags. The author believed this debug flag was required to circumvent a bug in the code/compiler/OS but, since it is required for an I/O routine, the impact on performance was not significant and, as such, we did not waste time locating the bug and simply employ the flag for only the routine in question.

2.2 Version 1.2.2

We found the User Guide for v1.2.0 employs CAM5.0; however, the links to the trusted output were either broken, or pointed to data sets now withdrawn, or irrelevant to v1.2.2.

Further, we found that the CESM 1.2.2 User Guide contains a number of errors, and the code itself requires a work-around to run with the Intel14 compilers available on ARCHER. Indeed, considerable effort was spent to investigate switching to the Intel13 compilers to work-around Intel14 internal compiler bugs; but this was circumvented by introducing a patch to CESM 1.2.2 made available directly to the author by a CESM developer.

3 Optimizations

3.1 Compiler flags

Once the given default versions of both v1.0.6 and v1.2.2 were installed and had passed our Port Validation process, we investigated a number of different optimization flags. This consumed both effort and cycles as used Task 1 and Task 2 from our Port Validation process to ensure the results remained correct, as the introduction of some optimization flags can generate incorrect results.

The result of this extensive piece of work gave the optimised flags were "-O3 -ftz –traceback". We also updated the debugging flags, replacing "–O2" with "-O0 -g -check uninit -check bounds -check pointers -fpe0".

NB: the debugging flags –ftz and –traceback were found to be necessary to pass our Port Validation for both the optimised and debugging sets of flags.

3.2 Parallel I/O

Lustre filesystem

By employing parallel NetCDF, we are able to exploit the Lustre file system on ARCHER, as Lustre is able to perform parallel I/O efficiently.

The details are this are outwith the scope of this report; however, technical details on how to tune your environment to get best Lustre I/O performance can be found here [8].

Fundamentally, Lustre achieves performance by storing a single file across multiple disks; this is called striping [8]. The number of "Object Storage Targets", or OSTs, employed affects how many disks a file is striped over. As CESM writes very large files, we ensure the target directories employ all OSTs as are available at that time, via the command

>lfs setstripe –c -1 target_directory_name

This ensures that all files and directories subsequently created within this directory inherit the same number of OSTs as the target directory and, as such, improve the performance of CESM's parallel I/O.

ARCHER modules

At the time, the optimisation of 1.2.2 revealed that ARCHER's parallel-netcdf module environment required updating for Intel compilers. Firstly, following the ARCHER User Guide, we attempted to use the module parallel-netcdf, however, this module stated that users should use the cray-parallel-netcdf module instead. The default version of this latter module contained a bug when using the Intel programming environment. The then most recent version did not contain this bug, but did require the user to avoid the default versions and employ the most recent versions of a collection of

modules, namely craype, cray-mpich, cray-netcdf, cray-hdf5 and cray-libsci. It should be noted that this inconsistency has since been fixed in the Module Updates, as of the 8th of October, 2014.

4 Performance Results

When performing timing experiments, each case was run at least three times, and the fastest time recorded.

4.1 Scaling results

In this subsection, we look at the performance results of both the small cases, case 7 and 8 from Task 1, can the two large cases, Case1 and Case2.

For v1.0.6 we considered Task1: case 7, named ERS.f19_f19.F, and Task 1: case 8, named ERS.f19_g16.I; and for v1.2.2, we considered Task 1: case 8, named ERS.f19_g16.I.

As previously stated, the two large cases are Case1, which has a resolution of f19_f19 and a *compset* of F_AMIP_CAM5, and Case 2 has a resolution f19_g16 and a compset of B_1850_CAM5_CN.

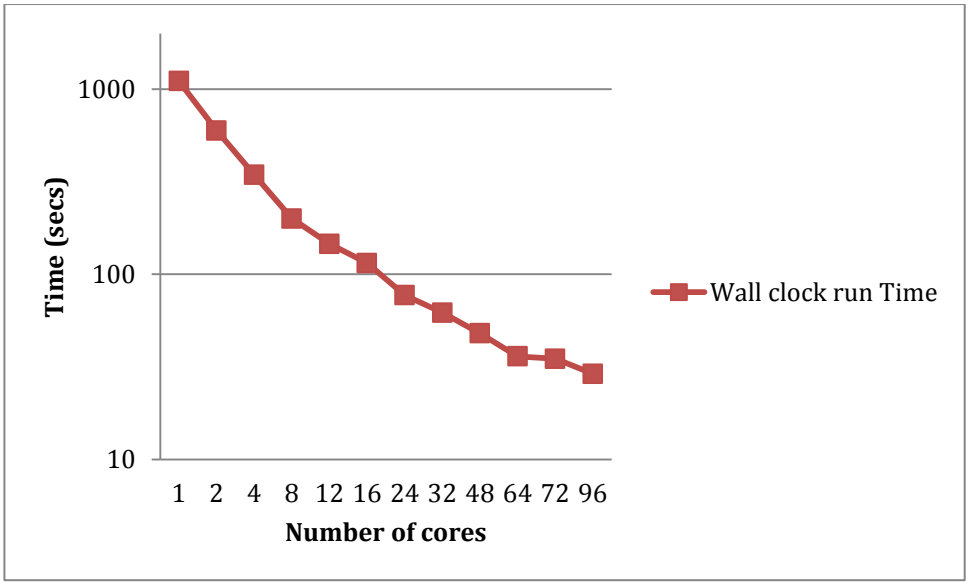

Fig. 1: Execution times for v1.0.6, Task1: case 7

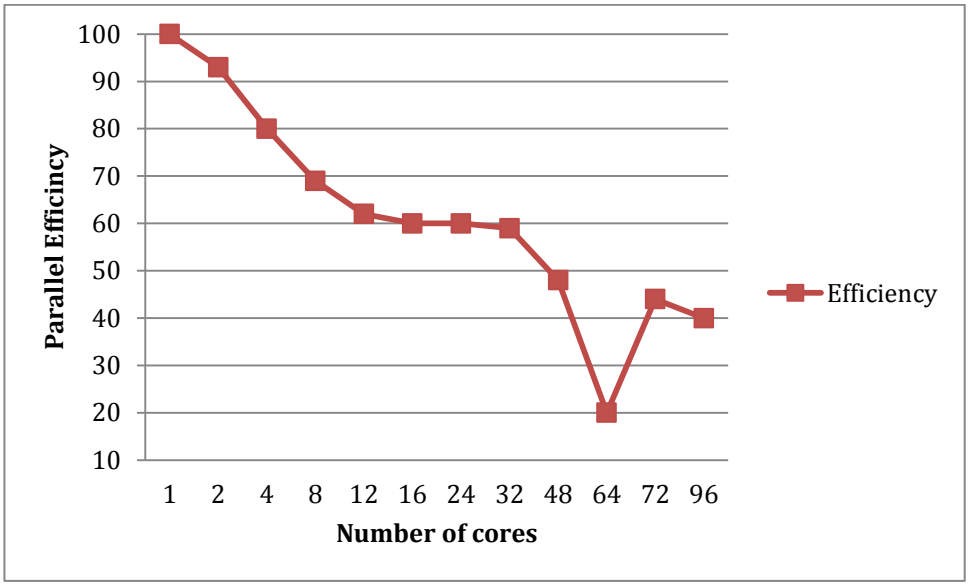

Fig. 2: Parallel Efficiency for v1.0.6, Task1: case 7

For v1.0.6, Task1: case7, ERS.f19_f19.F, the given default number of cores was set to 64 automatically by the installation process. We found the code scaled well; however, we saw a reduction in the parallel efficiency when using the default number of cores. For this case, on ARCHER, we recommend changing the number of cores to 48, e.g. two full nodes.

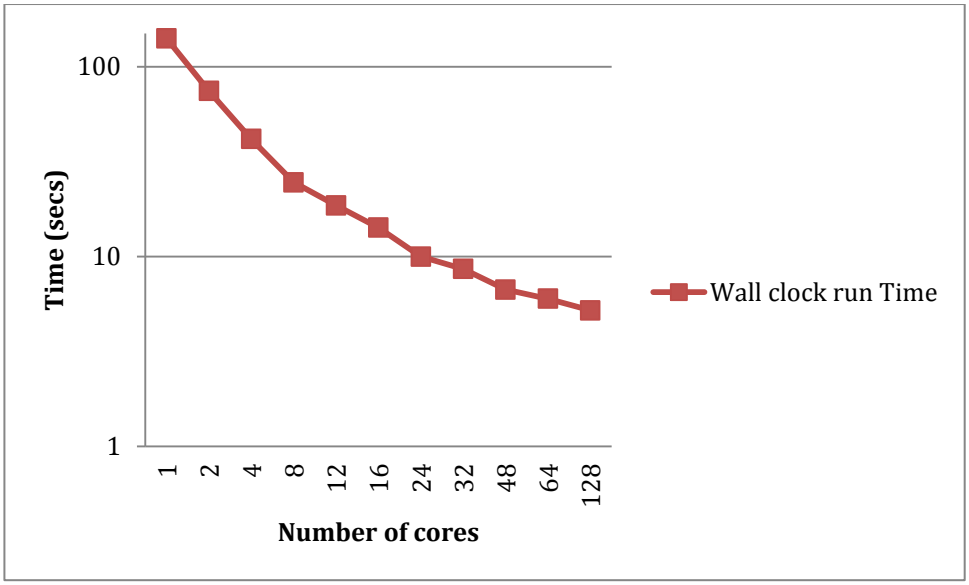

Fig. 3: Execution times for v1.0.6, Task1: case 8

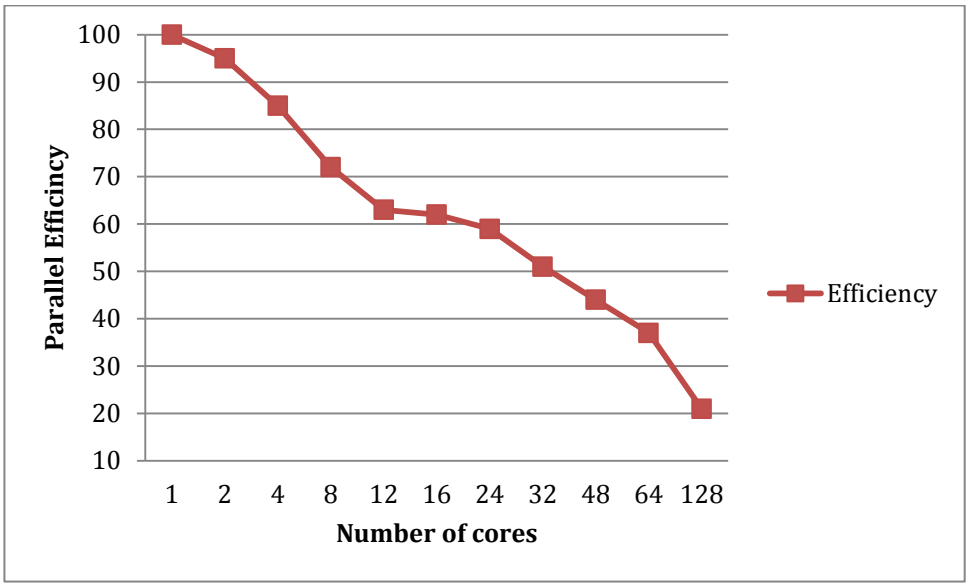

Fig. 4: Parallel Efficiency for v1.0.6, Task1: case 8

For v1.0.6, Task1: case8, ERS.f19_g16.I, the given default number of cores was also set to 64 automatically by the installation process. We found the code scaled well; however, we saw poor parallel efficiency when using the default number of cores. For this case, on ARCHER, we recommend changing the number of cores to 24.

For v1.2.2, Task1: case 8, ERS.f19_g16.I, the results are similar to v1.0.6 (results available from the author). The given default number of cores was also set to 64 automatically by the installation process. We found the code scaled well; however, we saw poor parallel efficiency when using the default number of cores. For this case, on ARCHER, we recommend changing the number of cores to 24, e.g. two full nodes.

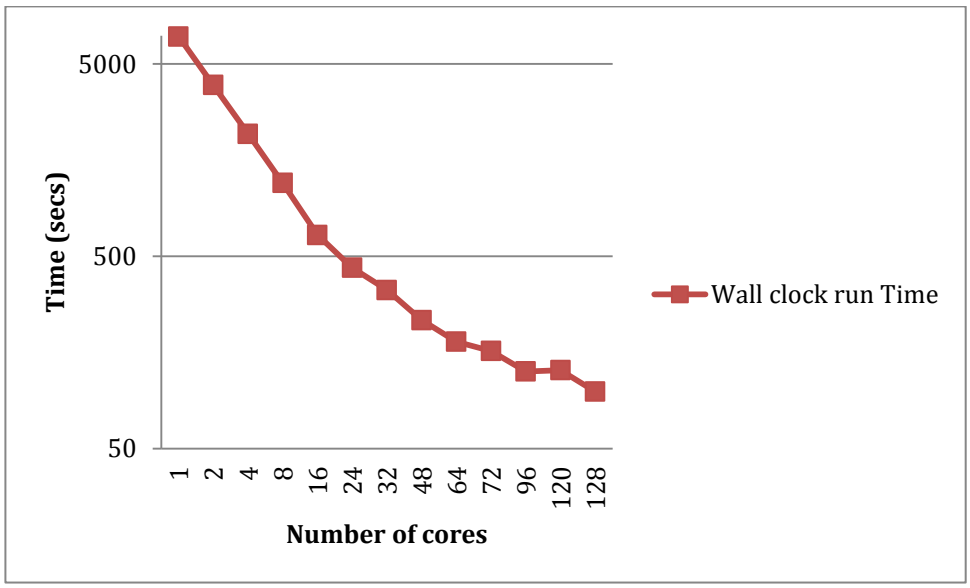

Fig. 5: Execution times for v1.0.6, large Case1

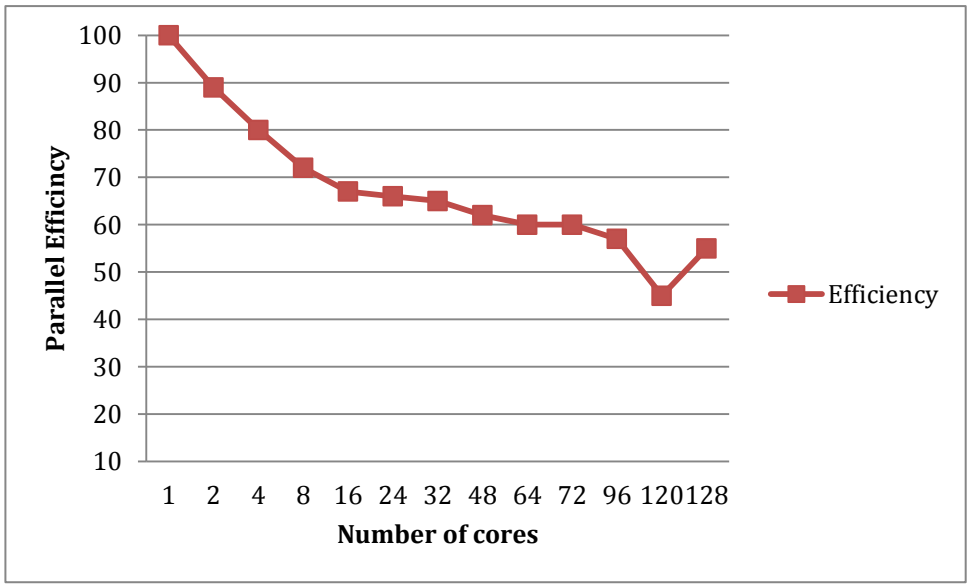

Fig. 6: Parallel Efficiency for v1.0.6, large Case 1

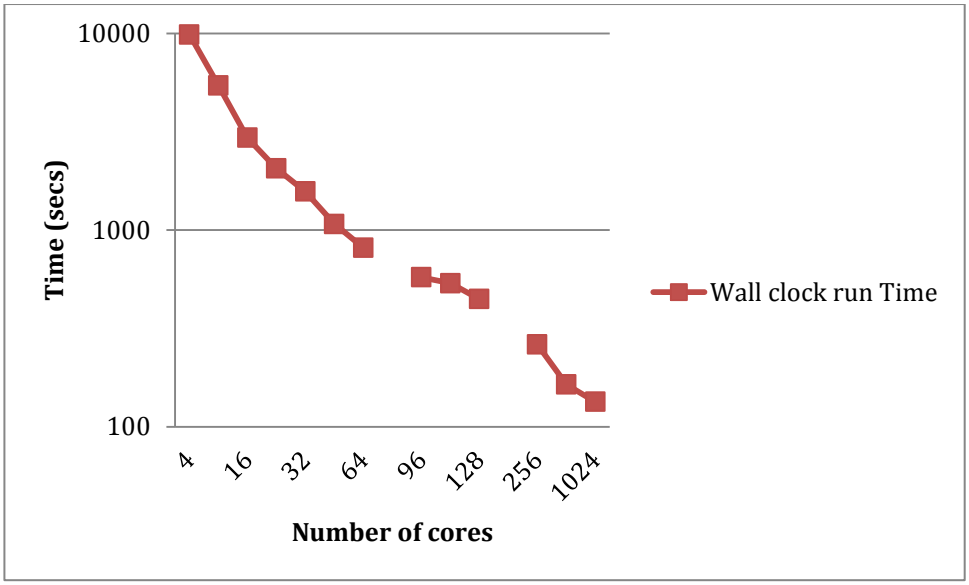

Fig. 7: Execution times for v1.0.6, large Case2

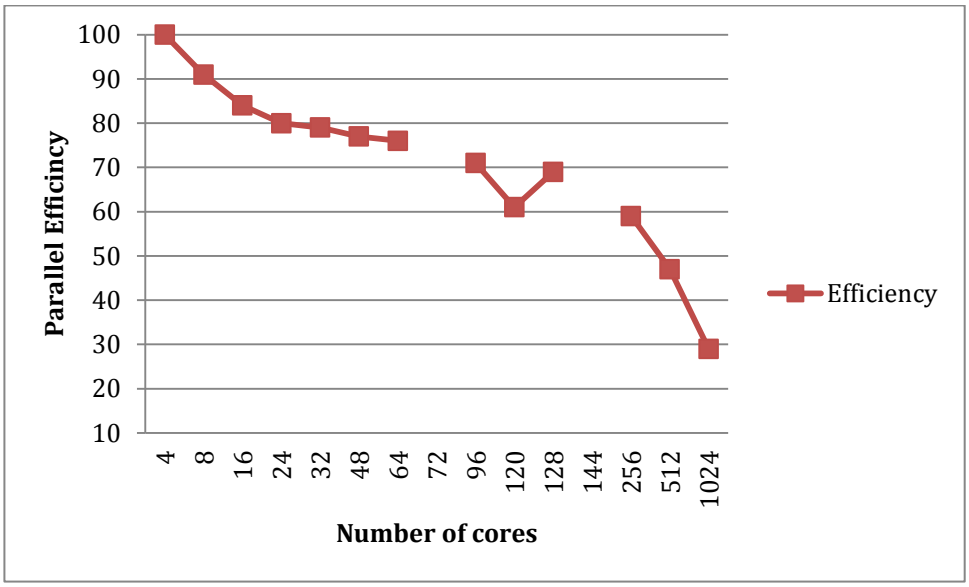

Fig. 8: Parallel Efficiency for v1.0.6, large Case 2

The results of timing Case 1 and Case 2 for v1.0.6 are presented in Figures 5 and 6, and 7 and 8. The results of timing exercise for Case 1 and Case 2 for v1.2.2 are similar (results available from the author).

For v1.0.6, Case 2 could not be run on ncores=1 or 2, due to a build failure (probably due to memory limitations), and on ncores=72 or

144, due to a configure failure (possibly due to a non-power of 2 number of cores)

For v1.0.6, the default number of cores for Case 1 and Case 2 was 64 and 128, respectively; however the most efficient number of cores was found to be 96 and 256, respectively.

Thus, for v1.0.6, the most efficient number of cores is smaller than the default for the small test cases, and larger for the two large cases.

This was the value employed when running Test 5, when running this case for a large 20 years simulation.

Both Case 1 and Case 2, which ran for 20 simulated years, produced correct answers. These simulations burned .1MAUs and .5MAUs, respectively.

For the large test cases, namely Case 1 and Case 2, for both v.1.0.6 and v1.2.2, the default number of cores for all four cases was given to be 64 cores.

After the extensive profiling exercise, we found the optimum number of cores on ARCHER was larger than the default value for v1.0.6 for both large cases: Case1: 128; Case2: 256; whereas, for v1.2.2, the default value of 64 was found to be the optimum number of cores. This suggests to the author that v1.2.2 of CESM calculates the default number of cores more accurately than v1.6.0.

Case 2 could not be run on ncores=1 or 2, due to a build failure (probably due to memory limitations), and on ncores=72 or 144, due to a configure failure (possibly due to a non-power of 2 number of cores)

The default number of cores for Case 2 was 128; however, the most efficient number of cores was found to be 256. This was the value employed when running Test 5, when running this case for a large 20 years simulation.

It was found that v1.0.6 appears to prefer powers-of-two core counts. However, it should be noted that ARCHER chargers each multiple of 24. Thus, when running on 32 cores, the user will be charged for 48 cores (~50% more); however, when running on 512 cores, say, the user is charged for 528 cores (~3% more). Thus, this is only of concern when running on low core counts.

When performing Task 4 of our Port Validation, we employed our optimal number of cores for v1.0.6, for both Case 1 and Case 2, where each ran for 20 simulated years. We found they produced correct answers. These simulations burned .1MAUs and .5MAUs, respectively.

4.2 Results from various methods of optimization

In this subsection, we consider the results of five different methods of optimization using v1.2.2 and the two large test cases.

The five methods examined where

1. the given default installation,
    a. serial netdf library, default OST value of 1
    b. referred to as ‘netcdf, ost=1’
2. the installation using optimisted compiler flags
    a. serial netcdf library, default OST value of 1, -O3
    b. referred to as ‘netcdf, ost=1, -O3’
3. default installation with parallel netcdf enabled
    a. parallel netcdf library, default OST value of 1
    b. referred to as ‘pnetcdf, ost=1’
4. default installation with parallel netcdf enabled
    a. parallel netcdf library, optimised OST value of -11
    b. referred to as ‘pnetcdf, ost=-1’
5. the installation using optimised compiler flags with parallel netcdf enabled
    a. parallel netcdf, optimised OST value of -1, -O3
    b. referred to as ‘pnetcdf, ost=-1, -O3’

For each of the five methods, various number of cores were used and each test was ran three times, and the fastest time reported.

The default number of cores for both Case 1 and Case 2 for v1.2.2 was given as 64. Once all the results were tabulated for all five methods, it was found that 64 was indeed the most efficient number of cores for all cases, except method 5, ‘pnetcdf, ost=-1, -O3’, where the optimum number of cores was found to be 96 and 72 for Case 1 and Case 2, respectively. (These values for the number of cores employed when running Test 5 of our Port Validation test, namely running both Case 1 and Case 2 for 20 years.)

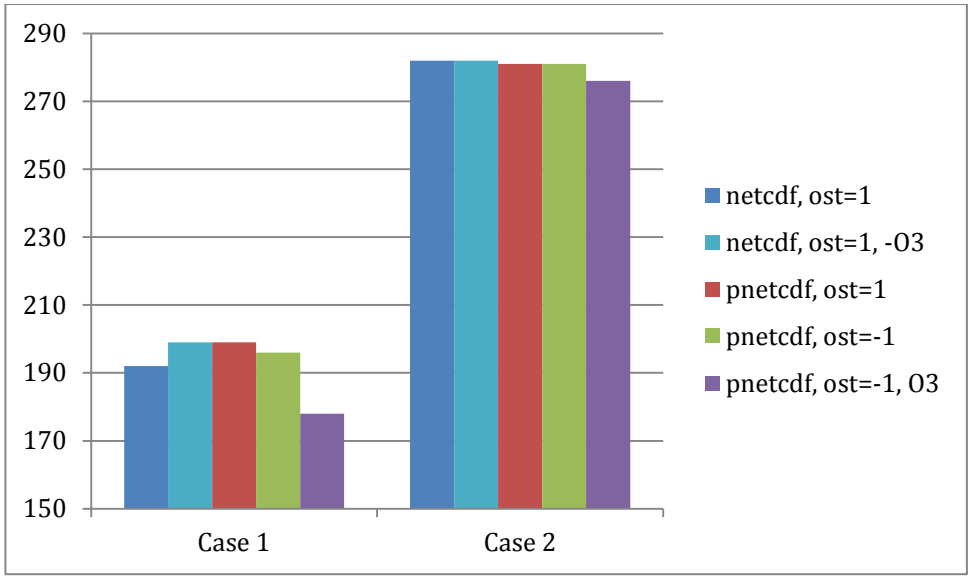

Fig. 9: Execution times for v1.2.2 running Case 1 and Case 2 and the 5 methods of optimization using the most optimal number of cores for each test.

From Figure 9 we can see that for Case 2, the time is reduced as the quality of optimsation method improves; however, for Case 1, it appears that when the optimised set of compiler flags is enabled without parallel I/O, the performance is reduced.

	Case 1	Case 2
time for optimised	178	276
time for default	192	282
percentage improvement	7.29	2.13

Table 1: v1.2.2 performance improvement

As can be seen in Table 1, using the optimised version of CESM v1.2.2, we have a speed up of 7.29% for Case 1 and 2.13% for Case 2.

5 Conclusions

Now that CESM is readily available, we envisage that there will be further growth in interest in the model. This state-of-the-art model complements those already planned for deployment on ARCHER, namely the Met. Office's Unified Model, WRF and WRF-CHEM in that:

1. CESM simulates tropical climate better than current alternative models;

2. CESM is a prominent model developed by the US climate research community. There is the need to facilitate the comparison between US and UK state-of-the-art models in order to resolve uncertainties in climate projections;

3. CESM can exploit the high level of parallelism provided by ARCHER;

4. CESM can be configured in many ways, opening a wide range of avenues for climate research.

5.1 Computational and Scientific Benefits

This state-of-the-art model complements those already planned for deployment on ARCHER, the Met. Office's Unified Model, WRF and WRF-CHEM, in that:

1. CESM simulates tropical climate better than current alternative models. The tropics represent an important area

of research for the climate scientific community. One motivation for this proposal comes from collaboration in this area between Edinburgh and Reading universities.

2. CESM is a prominent model developed by the US climate research community. There is the need to facilitate the comparison between US and UK models, given current models' complexity and in order to improve our understanding of their biases.

3. CESM can exploit the high level of parallelism provided by ARCHER, potentially permitting modelling at a relatively high resolution in reasonable wall-clock times.

4. CESM can be configured in many ways – to explore different models and parameterizations of land, sea, ice, atmosphere, at various spatial resolutions – opening a wide range of avenues for climate research.

An example of how the above factors combine together is in our intention to collaborate with UK partners in assessing the impact of increased atmospheric aerosols from human activities on regional climate variability and change. An early challenge is to better understand the changes in the Asian monsoon. Aerosols represent the major uncertainty in estimating climate sensitivity to increased greenhouse gases, and thus, hinder robust projections of future climate change.

The model therefore offers a significant new capability to the UK climate modelling community.

5.2 Lessons Learned

Use both the PDF and HTML versions of User Guides. The PDF version is the whole of the guide and is searchable, but may not include hot links, visible only in the HTML versions, to further important information.

Often it is desirable to install two versions of a particular code, namely the most up-to-date version and an older yet more popular version, where the latter is often incompatible with the former. It is typical that the older more familiar version is installed first; however, we would strongly suggest that the more up-to-date version is installed first. Thus, if there are any installation issues, the developers are more inclined to offer assistance as the newest version will have fixed bugs present in the older.

5.3 AMWG diagnostics tool

Lastly, the AMWG diagnostics package produces over 600 plots and tables from CCSM (CAM) monthly netcdf files [10].

For added value, the AMWG diagnostics tool is has also been ported to ARCHER [11].

5.4 Future Work

Both CESM 1.0.6 and 1.2.2 on ARCHER will be kept up-to-date when required. In other words, if users find the current set of instructions have become stale, e.g., they do not accurately describe the current installation process, then the instructions require updating. As such, users are kindly asked to submit an email to helpdesk@archer.ac.uk describing the required updates.

5.5 Acknowledgements

The codes in this project have already been ported to Edison at LBNL, where Edison is a similar machine to ARCHER. As such, this project has established a close and successful collaboration with the member of the CESM team responsible for the Edison port, and this has proven very successful when addressing bugs in both the CESM code and its associated documentation.

Special thanks are due to Jim Edwards, CESM Software Engineer at the National Centre for Atmospheric Research, Bolder, Colorado.

This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk).

5.6 References

[1]www.cesm.ucar.edu/modesl/cesm1.0

[2]www.archer.ac.uk

[3]www.archer.ac.uk/documentation/software/cesm/cesm106.php

[4]www.archer.ac.uk/documentation/software/cesm/cesm122.php
[5]www.cesm.ucar.edu/models/cesm1.2/cesm/doc/modelnl/machines.html
[6]www.cug.org/proceedings/cug2013_proceedings/includes/files/pap156.pdf
[7]www.hector.ac.uk/
[8]www.cesm.ucar.edu/models/cesm1.0/cesm/cesm_doc_1_0_6/x2323.html
[9] Performance of Parallel IO on ARCHER, D. Henty, A. Jackson, C. Moulinec, V. Szeremi, ARCHER white paper, June, 2015, www.archer.ac.uk/documentation/white-papers/parallelIO/ARCHER_wp_parallelIO.pdf
[10]www2.cesm.ucar.edu/working-groups/amwg/amwg-diagnostics-package
[11]www.archer.ac.uk/documentation/software/cesm/amwg.php

6 Appendix

The following subsection is quoted from the CESM User Guide and may be useful for users new to CESM.

6.1 For New CESM Users

Porting to Machines: Supported, Prototype and Generic

Scripts for supported machines, prototype machines and generic machines are provided

Supported machines have machine specific files and settings added to the CESM1 scripts. To get a machine ported and functionally supported in CESM, local batch, run, environment, and compiler information must be configured in the CESM scripts.

Prototype machines are machines in the CESM user community that CESM has been ported to and the machine specific files and settings were provided by the user community. Prototype machines all start with the prefix prototype_. These machines may not work out-of-the-box.

Generic machine generally refers more to classes of machines, like IBM AIX or a linux cluster with an intel compiler, and the generic machine names all start with the generic_ prefix. Generic machines require that a user provide some settings via command line options with create_newcase and then some additional effort will generally be required to get the case running. Generic machines are handy for

quickly getting a case running on a new platform, and they also can accelerate the porting process.

For more information on porting, see Chapter 7.

To get a list of current machines which are either Supported, Prototype or Generic, run script create_newcase with option -list from the $CCSMROOT directory.

The list of available machines are documented in CESM machines.

Running create_newcase with the "-list" option will also show the list of available machines for the current local version of CESM1.

Downloading input data

CESM input data will be made available through a separate Subversion input data repository.

DO NOT try to download the entire dataset.

CESM provides tools to check and download input data automatically.

A local input data directory should exist on the local disk, and it also needs to be set in the CESM scripts via the variable $DIN_LOC_ROOT_CSMDATA.

For supported machines, this variable is preset. For generic machines, this variable is set as an argument to create_newcase.

Multiple users can share the same $DIN_LOC_ROOT_CSMDATA directory. The files in the subdirectories of $DIN_LOC_ROOT_CSMDATA should be writeprotected. The directories in $DIN_LOC_ROOT_CSMDATA should generally be group writable, so the directory can be shared among multiple users.

When generating CESM executable, the utility, check_input_data is called, which tries to locate all required input. If required data is not found in $DIN_LOC_ROOT_CSMDATA, then the data will be downloaded automatically or downloaded by invoking check_input_data with the -export command argument. If you want to download the input data manually you should do it before you build CESM.

Quick Start (CESM Workflow)

The following quick start guide is for versions of CESM that have already been ported to the local target machine.

If CESM has not yet been ported to the target machine,
please see Chapter 7. If you are new to CESM1, please consider reading the introduction first

These definitions are required to understand this section:

• $COMPSET refers to the component set.

• $RES refers to the model resolution.
• $MACH refers to the target machine.
• $CCSMROOT refers to the CESM root directory.
• $CASE refers to the case name.
• $CASEROOT refers to the full pathname of the root directory where the case ($CASE) will be created.
• $EXEROOT refers to the executable directory. ($EXEROOT is normally NOT the same as $CASEROOT).
• $RUNDIR refers to the directory where CESM actually runs. This is normally set to $EXEROOT/run.

This is the procedure for quickly setting up and running a CESM case.

1. Download CESM (see Download CESM).

2. Select a machine, a component, and a resolution from the list displayed after invoking this command:
> cd $CCSMROOT/scripts
> create_newcase -list
See the component set table for a complete list of supported compset options.
See the resolution table for a complete list of model resolutions.
See the machines table for a complete list of machines.

3. Create a case.
The create_newcase command creates a case directory containing the scripts and xml files to configure a case (see below) for the requested resolution, component set, and machine. create_newcase has several

required arguments and if a generic machine is used, several additional options must be set (invoke create_newcase -h for help).
If running on a supported machine, ($MACH), then invoke create_newcase as follows:

```
> create_newcase -case $CASEROOT \
-mach $MACH \
-compset $COMPSET \
-res $RES
```

If running on a new target machine, see porting in Chapter 7.

4. Configure the case.
Issuing the configure command creates component namelists and machine specific
build and run scripts. Before invoking configure, modify the case settings in
$CASEROOT as needed for the experiment.

a. cd to the $CASEROOT directory.

```
> cd $CASEROOT
```

b. Modify configuration settings in env_conf.xml and/or in
env_mach_pes.xml (optional). (Note: To edit any of the env xml files, use
the xmlchange command. invoke xmlchange -h for help.)

c. Invoke the configure command.

```
> configure -case
```

5. Build the executable.

a. Modify build settings in env_build.xml (optional).

b. Run the build script.
> $CASE.$MACH.build

6. Run the case.

a. Modify runtime settings in env_run.xml (optional). In particular, set the DOUT_S variable to FALSE.

b. Submit the job to the batch queue. This example uses a submission command for a Cray computer:
> qsub $CASE.$MACH.run

7. When the job is complete, review the following directories and files

a. $RUNDIR. This directory is set in the env_build.xml file. This is the location where CESM was run. There should be log files there for every component (ie. of the form cpl.log.yymmdd-hhmmss). Each component writes its own log file. Also see whether any restart or history files were written. To check that a run completed successfully, check the last several lines of the cpl.log file for the string " SUCCESSFUL TERMINATION OF CPL7-CCSM".

b. $CASEROOT/logs. The log files should have been copied into this directory if the run completed successfully.

c. $CASEROOT. There could be a standard out and/or standard error file.

d. $CASEROOT/CaseDocs. The case namelist files are copied into this directory from the $RUNDIR.

e. $CASEROOT/timing. There should be a couple of timing files there that summarize the model performance.

f. $DOUT_S_ROOT/$CASE. This is the archive directory. If DOUT_S is FALSE, then no archive directory should exist. If DOUT_S is TRUE, then log, history, and restart files should have been copied into a directory tree here.

6.2 Employing CESM 1.2.2 on ARCHER

CESM 1.2.2 User Guide

The User Guide can be found at http://www.cesm.ucar.edu/models/cesm1.2/cesm/doc/usersguide/book1.html

NB: The PDF version is useful for searching the entire guide, however the PDF version gives no clue as to what text may be presented as links in the HTML version, so it is recommended to use both.

Installing CESM 1.2.2 on ARCHER

Firstly, edit your ~/.bashrc file and append the following lines

```
export CRAYPE_LINK_TYPE=dynamic
module load cmake
module load svn
module swap PrgEnv-cray PrgEnv-intel
module load cray-netcdf/4.3.2
module load cray-parallel-netcdf/1.4.1
module load cray-hdf5/1.8.13
```

These lines are required for each login session and batch job, thus placing them in the ~/.bashrc file will ensure the user does not forget to run them. This code requires the intel14 compiler which, in turn requires specific versions of craype, cray-parallel-netcdf, etc.

Download CESM 1.2.2 into your /work directory using svn, and then by following the instructions onhttp://www.cesm.ucar.edu/models/cesm1.2/tags/index.html#CESM1_2_2

Once downloaded, add the following 5 files into the 'scripts/ccsm_utils/Machines' directory.

122_config_machines.xml
env_mach_specific.archer
Depends.intel
122_config_compilers.xml
122_mkbatch.archer

NB rename 122_mkbatch.archer to mkbatch.archer, 122_config_compilers.xml to config_compilers.xml, and 122_config_machines.xml to config_machines.xml (i.e remove the first 4 characters)

Note, *before* building CESM, the input, archive and scratch directories must exist. The input directory already exists on ARCHER, and resides in a space which *any* ARCHER user can read and write to. The archive and scratch directories, on the other hand, must be created by hand by each user in their own work directory, e.g.

```
mkdir /work/ecse0116/ecse0116/gavin2/cesm1_2_2/archive
mkdir /work/ecse0116/ecse0116/gavin2/cesm1_2_2/scratch
mkdir /work/ecse0116/ecse0116/gavin2/cesm1_2_2/inputdata
```

NB if you will use parallel-netcdf and not simply netcdf then to gain best performance, you should set the LFS stripe to -1 for these three directories using the following three commands

```
lfs setstripe -c -1 /work/ecse0116/ecse0116/gavin2/cesm1_2_2/SCRATCH
lfs setstripe -c -1 /work/ecse0116/ecse0116/gavin2/cesm1_2_2/archive
lfs setstripe -c -1 /work/ecse0116/ecse0116/gavin2/cesm1_2_2/inputdata
```

These directories are then referenced in config_Machines.xml, e.g.

```
<DIN_LOC_ROOT>/work/ecse0116/ecse0116/gavin2/cesm1_2_2/inputdata</DIN_LO
C_ROOT>
<DIN_LOC_ROOT_CLMFORC>/work/ecse0116/ecse0116/gavin2/cesm1_2_2/ccsm1/in
putdata/atm/datm7</DIN_LOC_ROOT_CLMFORC>
<DOUT_S_ROOT>/work/ecse0116/ecse0116/gavin2/cesm1_2_2/archive/$CASE</DO
UT_S_ROOT>
<CESMSCRATCHROOT>/work/ecse0116/ecse0116/gavin2/cesm1_2_2/scratch</CES
MSCRATCHROOT>
```

Building the cprnc tool

Finally, one must build, by hand, the cprnc tool.

To make the cprnc tool, first upload the following file Makefile.cprnc122.archer to your cprnc directory, which will resemble:

/work/ecse0116/ecse0116/gavin2/cesm1_2_2/tools/cprnc

Once uploaded, run the following commands to make the cprnc tool (if they are not present in your ~/.bashrc file):

```
export CRAYPE_LINK_TYPE=dynamic
module load cmake
module load svn
module swap PrgEnv-cray PrgEnv-intel
module load cray-netcdf/4.3.2
module load cray-parallel-netcdf/1.4.1
module load cray-hdf5/1.8.13
```

and then copy over a file strangely missing from the down

```
cp ../../models/csm_share/shr/dtypes.h .
```

and then make the executable using the following three commands. (The 2nd throws an error which is fixed by simply running the command again)

```
make realclean -f Makefile.cprnc122.archer
make -f Makefile.cprnc122.archer
make -f Makefile.cprnc122.archer
```

Once the cprnc executable has been built, you must then edit the config_machines.xml file and replace the existing value of of CCSM_CPRNC to point to the location of your new cprnc executable, e.g. the following line must be changed by hand from

```
CCSM_CPRNC="/work/ecse0116/ecse0116/gavin2/CESM1.0/models/atm/cam/tools/cprnc/cprnc"
```

to something similar to

```
CCSM_CPRNC="/work/ecse0116/ecse0116/gavin/cesm1_2_2/tools/cprnc/cprnc"
```

This was a temporary bug in the intel compiler which may case the cprnc tool to throw either of the following errors at runtime:

```
Fatal Error: This program was not built to run in your system.
Please verify that both the operating system and the processor support Intel(R) AVX, F16C and RDRAND instructions.
```

or

```
Please verify that both the operating system and the processor support Intel(R) F16C instructions
```

This can be fixed by running the following commands

```
module swap craype-ivybridge craype-sandybridge
make clean -f Makefile.cprnc122.archer
make -f Makefile.cprnc122.archer
module swap craype-sandybridge craype-ivybridge
```

Completing the configuration process

6.2.1.1 Tools directory

By default, the taskmaker.pl tool is found in the scripts/ccsm_utils/Machines directory; however, the code expects this tool to reside in the scripts/ccsm_utils/Tools directory. One workaround is to copy the tool to the expected directory, e.g.

```
cd scripts/ccsm_utils cp Machines/taskmaker.pls Tools/.
```

Furthermore, some CESM scripts are not, by default, executable. A simple work-around which ensures the Tools are executable is to run the following command

```
chmod 755 scripts/ccsm_utils/Tools/*
```

6.3 Building CESM

Firstly, Change directory to the scripts directory in the 'work' installation of CESM, e.g.

```
cd /work/ecse0116/ecse0116/gavin/CESM1.0/scripts
```

6.4 Building tests

The process of building the CESM tests is slightly different from building simulations.

For the test ERS_D.f19_g16.X, say, issue the following commands in the 'scripts' directory.

```
./create_test -testname ERS_D.f19_g16.X.archer_intel -testid t21
cd ERS_D.f19_g16.X.archer_intel.t21
./ERS_D.f19_g16.X.archer_intel.t21.test_build
```

At present, the output of these processes contains multiple instances of the following string. NB this 'error' can safely be ignored.

```
ModuleCmd_Switch.c(172):ERROR:152: Module 'PrgEnv-cray' is currently not loaded
```

Running the test

To run the test, run the following command:

```
./ERS_D.f19_g16.X.archer_intel.t21.submit
```

6.5 Building your simulation

The code is built, from scratch, for each simulation the user wants to run.
Consider, say, the following model: f19_g16 B_18050_CAM5_CN

This is configured, built and submitted, for a case called, my_first_sim, say, using the following commands:

```
./create_newcase -case my_first_sim -res f19_g16 -compset B_1850_CAM5_CN -
mach archer -compiler intel
cd my_first_sim
./cesm_setup
./my_first_sim.build
```

Consider the create_newcase command: the -case flag assigns a local name to be given. Here I have used 'my_first_sim'; the -res flag assigns the mesh resolution; the -compset flag assigns the computation set of codes to employ; the -mach flag assigns the name of the platform; in this case 'archer'; and finally the -compiler flag assigns the name of the compiler; in this case 'intel' (which will employ intel14).

Consider the build command: if the input/restart files are not present, then the build command down loads the necessary files. As such, this command can take over an hour. Further, if the build fails with an error which references the /tmp directory, simply run the build command again as it is likely the system was very busy and the build command temporarily ran out of memory.

Before running the simulation, users should check both the your_name_for_this.archer.run file and the env_run.xml file, as the default values produce only a short run.

6.5.1.1 Running your simulation

To run the simulation, run the following command:

```
./my_first_sim.submit
```

6.6 How to change from the default settings

Before building

6.6.1.1 Changing the number of cores

Changing the number of cores to 128, say

```
cd $CASE
NTASKS=128
./xmlchange -file env_mach_pes.xml -id NTASKS_ATM -val $NTASKS
./xmlchange -file env_mach_pes.xml -id NTASKS_LND -val $NTASKS
./xmlchange -file env_mach_pes.xml -id NTASKS_ICE -val $NTASKS
./xmlchange -file env_mach_pes.xml -id NTASKS_OCN -val $NTASKS
./xmlchange -file env_mach_pes.xml -id NTASKS_CPL -val $NTASKS
./xmlchange -file env_mach_pes.xml -id NTASKS_GLC -val $NTASKS
./xmlchange -file env_mach_pes.xml -id NTASKS_ROF -val $NTASKS
./xmlchange -file env_mach_pes.xml -id NTASKS_WAV -val $NTASKS
./xmlchange -file env_mach_pes.xml -id TOTALPES -val $NTASKS
./cesm_setup -clean
./cesm_setup
./*.clean_build
./*.build
```

6.6.1.2 Changing simulation units

```
cd $CASE
# change given STOP_OPTION value to nyears
./xmlchange -file env_run.xml -id STOP_OPTION -val nyears
# change given STOP_N to 20
./xmlchange -file env_run.xml -id STOP_N -val 20
# don't produce restart files at end
./xmlchange -file env_run.xml -id REST_OPTION -val never
# or *do* produce restart files at end
#./xmlchange -file env_run.xml -id REST_OPTION -val $STOP_N
./cesm_setup -clean
./cesm_setup
./*.clean_build
./*.build
```

Parallel netcdf library

The parallel and serial versions of the netcdf are both available within the default build on ARCHER.

The default setting is to employ the serial netcdf libraries.

To employ the parallel netcdf libraries, change directory to the $CASE and run

```
./xmlchange -file env_run.xml -id PIO_TYPENAME -val pnetcdf
```

which change the value of PIO_TYPENAME from netcdf to pnetcdf, *before* building. (This is contrary to the User Guide which states the value is changed *after* building)

The number of IO tasks is PIO_NUMTASKS, and the default value is -1 which instructs the library to select a suitable default value.

As stated above, if using parallel-netcdf and not simply netcdf, then to gain best performance, you should set the LFS stripe to -1 for your SCRATCH, archive and inputdata directories.

```
lfs setstripe -c -1 /work/ecse0116/ecse0116/gavin2/cesm1_2_2/SCRATCH
lfs setstripe -c -1 /work/ecse0116/ecse0116/gavin2/cesm1_2_2/archive
lfs setstripe -c -1 /work/ecse0116/ecse0116/gavin2/cesm1_2_2/inputdata
```

Changing the batch script

6.6.1.3 Change the budget to your budget account

In the file mkbatch.archer, change the line

```
set account_name = "ecse0116"
```

to

```
set account_name = "<budget>"
```

where the string <budget> is replaced by the name of your budget on ARCHER.

6.6.1.4 Editing the batch script

The batch script is a file which ends with '.run', thus to edit the batch script using vi, say, type the following

```
vi *.run
```

6.6.1.5 Requesting high memory nodes

Archer has two types of compute nodes, 2632 nodes with 64MBs of shared memory and 376 nodes with 128MBs. Both have 24 cores which share this memory.

During the Validation Process, it was found that the larger memory nodes were required to run some of the tests. To use the larger memory nodes, update the batch script, namely the *.run file, to select the larger memory nodes, specifically, to run one 4 large memory nodes, set

```
#PBS -l select=4:bigmem=true
```

else to run on 4 smaller memory nodes set

```
#PBS -l select=4:bigmem=false
```

or, if you don't mind which node you run on, set

```
#PBS -l select=4
```

6.6.1.6 Requesting longer wall times

Users are limited to requesting a maximum wall time of 24 hours, e.g.

```
#PBS -l walltime=24:00:00
```

however, if the job requires more time, then users can increase this limit to 48 hours by using the PBS long flag, e.g.

```
#PBS -q long
#PBS -l walltime=48:00:00
```