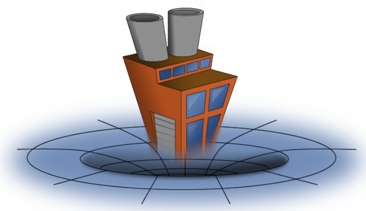
# Setting up Simulation Factory on NCAR Cheyenne

## Simulation Factory introduction



Simulation Factory (https://simfactory.org), or "simfactory" for short, is the Einstein Toolkit's tool to handle compiling Cactus on compute clusters and laptops. It also provides functionality to create and manage simualtions, and trys to provide a common interface for clusters using different resource managers.

In the word of the simfactory authors:

> Performing large three-dimensional time-dependent simulations is a complex numerical task. Managing such simulations, often several at the same time as they execute on different supercomputers, is comparable to herding cats — supercomputers differ in their hardware configuration, available software, directory structure, queueing systems, queuing policies, and many other relevant properties.
> However, these differences are only superficial, and the basic capabilities of supercomputers are very similar. The simulation factory contains a set of abstractions of the tasks which are necessary to set up and successfully finish numerical simulations using the Cactus framework. These abstractions hide tedious low-level management tasks, they capture "best practices" of experienced users, and they create a log trail ensuring repeatable and well-documented scientific results. Using these abstractions, many types of potentially disastrous user errors are avoided, and different supercomputers can be used in a uniform manner.

Today's simfactory is simfactory version 2 (https://arxiv.org/abs/1008.4571), a Python 3 application re-implementing the functionality of Erik Schnetter's original Perl based simfactory code.

### Simfactory directory structure

Users typically interact with simfactory using its command line interface via the `sim` command wrapper. For example `sim build --thornlist einsteintoolkit.th etk` compiles a Cactus configuration `etk` using the thorn list file `einsteintoolkit.th` .

This command is identical on all systems supported by simfactory. How then does simfactory know which compiler to use and which options to pass to the Cactus build system (and latter, how to submit a simulation to the cluster's resource manager)?

Simfactory contains an extensive "machine database" in the `mdb` subdirectory that contains information specific to each cluster. A complete entry for a machine consistents of 4 files in subdirectories of `mdb` :

- a machine "ini" file in the `machines` subdirectory. This file uses a MS Windows ini (https://en.wikipedia.org/wiki/INI_file) style ini syntax, to identify and describe a machine and ties all 4 files together. Options in machine ini files can be overwritten using section in the user's `etc/defs.local.ini` file in simfactory.
- a option list file in the `optionlists` directory. This is a standard Cactus configuration file as described in chapter B2.1 (http://einsteintoolkit.org/usersguide/UsersGuide.html#x1-19000B2.1) "Configuration Options" of the Cactus UsersGuide
- 2 template files, a submit sript in `submitscripts` and a run script in `runscripts` , that are used when submitting simulations to the resource manager. The former typically contains the resource manager comments (e.g. `#SBATCH -A MyAllocation` ) and is the template for the batch job script passed to the resource manager. The latter, the run script, is a bash script template file that contains the "meat" of the batch job script.

For example this is what a directory layout of files used to descripe a cluster "foo" could look like:

```
simfactory
  |
  +- mdb
  |   |
  |   +- machines/foo.ini
  |   +- optionlists/foo-gnu.cfg
  |   +- runscripts/foo-gnu.run
  |   +- submitscripts/foo.sub
  |
  +- etc
      |
      +- defs.local.ini
```

Setting up simfactory for a new machine means providing the 4 mdb files to simfactory.

This is, unfortunately, not a simple task and requires experience using clusters as well as some understanding of simfactory. In particular, setting up simfactory on a cluster is quite a bit more complex than "only" compiling and submitting a Cactus simulation using example batch job scripts and suggested environment modules (https://en.wikipedia.org/wiki/Environment_Modules_(software)) the cluster's help page. Simfactory does not provide any automation or help when incorporating a new machine.

Over the years a number of attempts were made to document this, first in the Simulation Factory User Guide (https://simfactory.bitbucket.io/simfactory2/userguide/) by Ian Hinder, then in the Simulation Factory Advanced Tutorial (https://docs.einsteintoolkit.org/et-docs/Simulation_Factory_Advanced_Tutorial) by Michael Thomas and Ian Hinder, and finally in the Configuring a new machine (https://docs.einsteintoolkit.org/et-docs/Configuring_a_new_machine) Einstein Toolkit wiki page by Ian Hinder and Roland Haas. All of these still provide valueable, if incomplete, sources of information that should be consulted by anyone attempting to set up simfactory on a new machine.

This tutorial finally, attempts to document my (Roland Haas) own steps to set up simfactory on a new machine, specifically the Cheyenne (https://arc.ucar.edu/knowledge_base/70549542) cluster at NCAR (https://arc.ucar.edu/).

## How to use this tutorial

This tutorial will not list every possible option available in simfactory, nor does it aim to replace simfactory's documentation. For thos please consult the Configuring a new machine (https://docs.einsteintoolkit.org/et-docs/Configuring_a_new_machine) Einstein Toolkit wiki and / or the Simulation Factory User Guide (https://simfactory.bitbucket.io/simfactory2/userguide/) and relevant chapter (http://einsteintoolkit.org/usersguide/UsersGuide.html#x1-19000B2.1) "Configuration Options" of the Cactus UsersGuide.

Instead it provides a narrated tour of how I set up simfactory on a new cluster, trying to provide helpful tips and point out pitfalls that you may encounter. As such: your milage may vary!

## Before you start: gather information

Setting up simfactory is not for the faint of heart, but a lot trouble can be avoid by spending some time gathering information. Setting up simfactory is also not fast, it takes me, whom I consider to be familiar with HPC systems and who has done this before, at least half a day for an initial setup and likely a couple of months of using the setup to iron out kinks. So if you have never added a machine to simfactory, have never used the machine in question before, and this is you first time using the Einstein Toolkit, you are in for a tough ride (but is has been done) and should expect to spent multiple days. However, help is available, usually from the Einstein Toolkit users group users@einsteintoolkit.org (mailto:users@einsteintoolkit.org) and from the cluster admins.

With this out of the way, these are information would collect:

- find the cluster's documentation website, e.g. https://arc.ucar.edu/knowledge_base/70549542 (https://arc.ucar.edu/knowledge_base/70549542)
  - locate information on how to log in, e.g. https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-LogginginonanNCARsystem (https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-LogginginonanNCARsystem)
  - locate information on what environment module system is used, e.g. https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-Environment (https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-Environment) You should be familiar with environment modules (https://lmod.readthedocs.io/en/latest/) and the `module avail` , `module spider` and `module load` commands.
  - locate information on the file systems on the cluster, you want one directory that does not purge to store source code and compile, and a scratch directory where to run simulations, e.g. https://arc.ucar.edu/knowledge_base/68878466 (https://arc.ucar.edu/knowledge_base/68878466)
  - locate information on how to compile a code which compilers to use etc., e.g. https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-compilingCompiling (https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-compilingCompiling) . Usually I start with the default choices made by the admins for C/C++/Fortran compiler and `MPI` library. Everything else is optional and can be ignored if needed. However you may want to take note of "typical" libraries for use by the ET for example `HDF5` , `GSL` , `BLAS` / `LAPACK` / `MKL`
  - find a "sample batch" script, e.g. https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-submittingSubmittingjobs (https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-submittingSubmittingjobs)
    - find out about which queues exist usually from the docs, e.g. https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-Cheyennequeues (https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-Cheyennequeues) but if all else fails, and assuming the system uses SLURM, you can use the output of `scontrol show partitions` or for PBS like systems `qstat -q` .
- download (and know how to compile and run) an MPI "Hello, world!" code, for example this (https://mpitutorial.com/tutorials/mpi-hello-world/) or even better Blue Waters's (https://bluewaters.ncsa.illinois.edu/liferay-content/document-library/Documentation%20Documents/aprun/hello_world.c) more informative one. This will be helpful to check that the resource manager is being used correclty later on

- test that the current development version of the Einstein Toolkit compiles fine on a known good system, e.g. your laptop. This way, if compilation fails. you kow it's not a (necessarily) a bug in the toolkit

## Getting started

First, log in to the machine. I assume this you know how to do, if not the cluster docs will explain how to do so:

```
> ssh rhaas@cheyenne.ucar.edu
(rhaas@cheyenne.ucar.edu) Token_Response:
Last failed login: Thu Feb 24 01:31:06 MST 2022 from 141.142.41.4 on ssh:no
tty
There was 1 failed login attempt since the last successful login.

***************************************************************************
***
*                  Welcome to Cheyenne - February 23, 2021
***************************************************************************
***
            Today in the Daily Bulletin (arc.ucar.edu)

             - Join NCAR HPC User Group (NHUG) monthly call on March 1, 9-1
0 a.m. MT

Documentation:        https://bit.ly/CISL-user-documentation
Key module commands:  module list, module avail, module spider, module help
CISL Help:            support.ucar.edu -- 303-497-2400
---------------------------------------------------------------------------
-----
```

On Cheyenne the `$HOME` directory is 50GB is size (https://arc.ucar.edu/knowledge_base/68878466) which is suitable for the ET source code tree. There are also `Work` directories (1TB) that could be used, or used to stage simuilation input data, and a `Scrach` directory to run simulations in (purged).

Since login drops be off in `$HOME` I can directly proceed to download (http://einsteintoolkit.org/download.html) a development copy of the ET:

```
> curl -kLO https://raw.githubusercontent.com/gridaphobe/CRL/master/GetComp
onents
> chmod a+x GetComponents
> ./GetComponents --parallel https://bitbucket.org/einsteintoolkit/manifes
t/raw/master/einsteintoolkit.th
```

If this fails since `git` or `svn` are not found then you most likely have to load a module for them. See the output of `module avail 2>&1 | less` for which modules to load.

Set up `git` on the machine so that you can commit changes to simfactory's git repo, typically:

```
> git config --global user.author "Roland Haas"
> git config --global user.email rhaas@illinois.edu
```

though `git` will tell you what to do when you first attempt to commit.

If this fails b/c the cluster has not access to the Internet (SuperMUC is like this), then you will have to download the ET on your laptop and use `rsync` to copy to the cluster.

Assuming all went well you should see the usual `GetComponents` output ending in:

```
    Checking out module: CactusExamples/WaveToy2DF77
        from repository: https://bitbucket.org/cactuscode/cactusexamples.git
                   into: Cactus/arrangements
  -----------------------------------------------------------------
    303 components checked out successfully.
    0 components updated successfully.

    Time Elapsed: 1 minutes, 11 seconds
```

With this we can already check if simfactory's machine database already contains a entry for the machine (unlikely) or confuses it with an existing machine. For this we use the `whoami` subcommand of simfactory:

```
> cd Cactus
> ./simfactory/bin/sim whoami
```

which should respond with something like:

```
Warning: Unknown machine name cheyenne6.cheyenne.ucar.edu
Warning: Unknown machine name cheyenne6.cheyenne.ucar.edu
Current machine: None
```

Importantly the machine should be reported as "unknown" and the current machine as `None` .

Take note of the machine name that simfactory reported, e.g. `cheyenne6.cheyenne.ucar.edu` in my case. We will use this to construct an `aliaspattern` entry in the machine ini file to identify the machine.

If simfactory reports an existing machine (but not the correct one), then we will have to tighten the alias pattern used by the offending machine. If, for example, `Current machine: golub` had been returned the `aliaspattern` entry for `golub` needs updating. You can find the correct ini file by `grep` ing for `golub` in all files in `simfactory/mbb/*.ini` . Note that the file *name * does *not* matter. What matters is the name of the (single) section in the file e.g. `[golub]` . Once you have found the file, you have to make a guess on how the alias pattern should be tightened. This may require login into that machine and checking its `hostname` . You can also set `aliaspattern` (temporarily) to and empty pattern `^$` which will never match and thus avoid the misidentification.

In the (likely) case of no conflict you now know the hostname of one of the login nodes and can start setting up machine ini file.

## Copying and editing files

You should however not create such a file from scratch, instead start by copying a "similar" files from a "similar" system. Typically you want to check that the queueing system is the same and also things like Cray/non-Cray system. It is also advisable to copy the machine ini file from an active, in use system and not a long-dead system ones.

Since Cheyenne uses the PBS queuing system (typically documented in the doc section on how to submit (https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-submittingSubmittingjobs) jobs, I want to use similar system. How to find this template system is not straightforward. Ideally you are familiar with the one to copy from already. If you know (e.g. from inspecting the sample batch job scripts (https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-submittingSubmittingjobs) provided in the documentation) what the comment leader ( `SBATCH` , `PBS` , etc.) is then you can use that to find candidates in the submitscripts.

In my case, `PBS` is the comment leader and using

```
> grep -l PBS simfactory/mdb/submitscripts/*.sub
```

which returns all run scripts containing `PBS` :

```
simfactory/mdb/submitscripts/bluewaters.sub
simfactory/mdb/submitscripts/golub.sub
simfactory/mdb/submitscripts/pbs-normal.sub
simfactory/mdb/submitscripts/philip-mpich.sub
simfactory/mdb/submitscripts/qb.sub
simfactory/mdb/submitscripts/shelob.sub
simfactory/mdb/submitscripts/smic.sub
simfactory/mdb/submitscripts/sunnyvale.sub
```

and I picked the `bluewaters` one as the one I am most familiar with. Make a copy:

```
> cp simfactory/mdb/submitscripts/bluewaters.sub simfactory/mdb/submitscrip
ts/cheyenne.sub
```

where the target file name `cheyenne.sub` is arbitrary but naming it after the "short name" of the machine is useful to identify files.

Simfactory uses placeholders of the from `@FOO@` in runscript and submitscripts to inject information about the run. These are (mostly) described in the Simulation Factory User Guide (https://simfactory.bitbucket.io/simfactory2/userguide/), or in the Simulation Factory Advanced Tutorial (https://docs.einsteintoolkit.org/et-docs/Simulation_Factory_Advanced_Tutorial), or in the Configuring a new machine (https://docs.einsteintoolkit.org/et-docs/Configuring_a_new_machine) Einstein Toolkit wiki page.

## Submitscript

Next edit the new using the sample batch job scripts to fill in all information required. In Cheyenne's case, and most likely for other clusters, I needed to ajust the lines dealing with node allocations: `-l nodes=...` in BlueWaters and `-l select` for Cheyenne. In the end I made some minor changes only:

```
> diff -u simfactory/mdb/submitscripts/bluewaters.sub simfactory/mdb/submit
scripts/cheyenne.sub
--- simfactory/mdb/submitscripts/bluewaters.sub 2021-08-31 09:57:09.5529590
00 -0600
+++ simfactory/mdb/submitscripts/cheyenne.sub   2021-09-16 13:04:11.9503380
00 -0600
@@ -1,12 +1,13 @@
 #! /bin/bash
 #PBS -A @ALLOCATION@
-#PBS -q @(ifthen("@QUEUE@"[-3:-1]==":x", "@QUEUE@"[:-3], "@QUEUE@"))@
-#PBS -r n
+#PBS -q @QUEUE@
 #PBS -l walltime=@WALLTIME@
-#PBS -l nodes=@NODES@:ppn=@PPN@:@("@QUEUE@"[-3:]==":xk" ? "xk" : "xe")@
+#PBS -l select=@NODES@:ncpus=@PPN@:mpiprocs=@NODE_PROCS@:ompthreads=@NUM_T
HREADS@
 #PBS @("@CHAINED_JOB_ID@" != "" ? "-W depend=afterany:@CHAINED_JOB_ID@" :
 "")@
 #PBS -N @SIMULATION_NAME@
 #PBS -m abe
+#PBS -M @EMAIL@
+#PBS -k eod
 #PBS -o @RUNDIR@/@SIMULATION_NAME@.out
 #PBS -e @RUNDIR@/@SIMULATION_NAME@.err
 cd @SOURCEDIR@
```

Ian Hinder's simfactory users guide (https://simfactory.bitbucket.io/simfactory2/userguide/processterminology.html) lists the conventions used for nodes, MPI ranks, threads, etc. in simfactory and is very handy for this.

| Variable | Option | Definition |
|---|---|---|
| NODES | | nodes |
| PROCS_REQUESTED | | cores |
| PPN | –ppn | cores per node |
| NUM_PROCS | | processes |
| NODE_PROCS | | processes per node |
| PROCS | –procs | threads |
| NUM_THREADS | –num-threads | threads per process |
| PPN_USED | –ppn-used | threads per node |
| NUM_SMT | –num-smt | threads per core |

I strongly advise to ignore any hyperthreading settings for now since these are usually cluster specific and not very useful for Cactus.

## Runscript

Runscript and submit script form a pair so typically one needs starts from the same cluster as a template. In my case BlueWaters.

```
> cp simfactory/mdb/runscripts/bluewaters.run simfactory/mdb/runscripts/che
yenne.run
```

the edit it in an editor. Since BlueWaters, being a Cray uses aprun but Cheynne uses (in my case) OpenMPI some changes are required to the actual Cactus command line.

Here consulting the cluster provided sample job scripts for a hybrid MPI+OpenMP job using the selected MPI stack are useful: https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-submittingSubmittingjobs (https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-submittingSubmittingjobs)

You may want to test out e.g. `mpirun` for the MPI stack to use:

```
> bash -i
> module load openmpi/4.0.5
> which mpirun
> mpirun --version
> exit
```

which uses a trick of starting a new shell, then loading the module in there. This ensures that once one runs `exit` the sub-shell quits and I do not change my actual module environment.

I picked some information out of other runscripts using OpenMPI ( `golub` ) to set all OpenMPI options and create a uniform job script.

### Run script diff

In end end I based my runscript on Golub's script since it also uses PBS and OpenMPI:

```
> diff -uw simfactory/mdb/runscripts/golub.run simfactory/mdb/runscripts/ch
eyenne.run
--- simfactory/mdb/runscripts/golub.run 2021-08-31 09:57:09.545304000 -0600
+++ simfactory/mdb/runscripts/cheyenne.run      2021-09-16 12:41:56.0430780
00 -0600
@@ -1,19 +1,22 @@
-#!/bin/sh
+#! /bin/bash

 echo "Preparing:"
 set -x                          # Output commands
 set -e                          # Abort on errors

+cd @RUNDIR@-active
+
 echo "Checking:"
 pwd
 hostname
 date
+cat ${PBS_NODEFILE} > SIMFACTORY/NODES

 echo "Environment:"
 export CACTUS_NUM_PROCS=@NUM_PROCS@
 export CACTUS_NUM_THREADS=@NUM_THREADS@
 export OMP_NUM_THREADS=@NUM_THREADS@
-#env | sort > SIMFACTORY/ENVIRONMENT
+env | sort > SIMFACTORY/ENVIRONMENT

 echo "Job setup:"
 echo "   Allocated:"
```

## Option list

Next I take a look at the option list, which is used to compile Cactus. This would have been easisest to do first actually since it does not require submit or run information.

Again, start from a working option list for a similar system. Here though the determining factor is what compiler and MPI stack to use. Sticking with the default compiler for now, one can check which modules are currently loaded:

```
> module list

Currently Loaded Modules:
  1) ncarenv/1.3   2) intel/19.1.1   3) ncarcompilers/0.5.0   4) mpt/2.22
5) netcdf/4.8.1
```

which shows the Intel compiler and HPE MPI stack mpt (the latter I did not know about and swapped for OpenMPI).

If you are unclear what a module does, then the `module show` command is useful. For example for `mpt` :

```
> module show mpt
--------------------------------------------------------------------------
------------------------
   /glade/u/apps/ch/modulefiles/default/intel/19.1.1/mpt/2.22.lua:
--------------------------------------------------------------------------
------------------------
family("mpi")
whatis("MPT MPI v2.22")
help([[The HPE Message Passing Interface (MPI) is an MPI development enviro
nment
designed to enable the development and optimization of high performance
computing (HPC) Linux® applications.

Software website - https://www.hpe.com/us/en/product-catalog/detail/pip.hpe
-performance-software-message-passing-interface.1010144155.html

Built on Wed May 27 16:02:29 MDT 2020
Modules used:
   intel/19.1.1
]])
prepend_path("PATH","/glade/u/apps/ch/opt/mpt/2.22/bin")
prepend_path("MANPATH","/glade/u/apps/ch/opt/mpt/2.22/man")
prepend_path("CPATH","/glade/u/apps/ch/opt/mpt/2.22/include")
prepend_path("FPATH","/glade/u/apps/ch/opt/mpt/2.22/include")
prepend_path("LD_LIBRARY_PATH","/glade/u/apps/ch/opt/mpt/2.22/lib")
prepend_path("LIBRARY_PATH","/glade/u/apps/ch/opt/mpt/2.22/lib")
setenv("MPI_ROOT","/glade/u/apps/ch/opt/mpt/2.22")
setenv("MPT_VERSION","2.22")
setenv("LMOD_MPI","mpt_fmods")
setenv("LMOD_MPI_VERSION","2.22")
setenv("MPICC_CC","icc")
setenv("MPICXX_CXX","icpc")
setenv("MPIF90_F90","ifort")
setenv("MPIF08_F08","ifort")
setenv("OSHCC_CC","icc")
setenv("OSHCXX_CXX","icpc")
setenv("OSHF90_F90","ifort")
prepend_path("CPATH","/glade/u/apps/ch/opt/mpt_fmods/2.22/intel/19.1.1/")
prepend_path("LD_LIBRARY_PATH","/glade/u/apps/ch/opt/mpt_fmods/2.22/intel/1
9.1.1/")
append_path("MODULEPATH","/glade/u/apps/ch/modulefiles/default/mpt/2.22/int
el/19.1.1")
prepend_path("PATH","/glade/u/apps/ch/opt/ncarcompilers/0.5.0/intel/19.1.1/
mpi")
setenv("NCAR_LIBS_MPT","-lrt -ldl")
setenv("NCAR_ROOT_MPT","/glade/u/apps/ch/opt/mpt/2.22")
setenv("NCAR_RANK_MPT","1000")
```

which will come in useful afterwards. The Intel compiler's executable name is `icpc` which is something to look for in existing option lists. Since I picked the `OpenMPI` stack out of the one `module avail` output I picked `shelob-openmpi.cfg` as the starting point, which I know uses the Intel compiler and `OpenMPI`. Otherwise `grep -l icpc` can be used similar to the submit script.

```
> cp shelob-openmpi.cfg cheyenne.cfg
```

Setting `CPP` , `FPP` , `CC` , `CXX` , `F90` is fairly straightforward when using the module system. A tricky one is that `FPPFLAGS` must be set (if `FPP` is set) to `--traditional` to avoid strange parsing errors in Fortran files that are being passed through `FPP` to expand C preprocessor macros.

Most other `<FOO>FLAGS` can be left alone usually or upated based on documentation on how to compile (https://arc.ucar.edu/knowledge_base/72581213#QuickstartonCheyenne-compilingCompiling). Required settings are `-std=gnu99` and `-std=gnu++11` for language support required by Cactus and `-safe_cray_ptr` to allow use of "Cray" pointers in Fortran code which is required by the ET ( `GRHydro` in particular).

The Intel compiler (all versions) rely on the C++ STL library shipped with the GNU compiler and need to be told which one to use if the admins did not already set things up correctly. Usually it is easiest to add a `-gxxname <FULL-PATH-TO-G++>` to `CXXFLAGS` to force this. gcc >= 6 should be sufficient for all needs of the ET. Note that not all version are supported, in particular too *new* versions can cause problems as well as too old ones.

`LIBS` also is tricky and often (has to) lists the Fortran runtime library ( `ifcore` for Intel, `gfortran` for GNU) so that the C++ linker includes it in our mixed language code.

When using the Intel compiler, make sure to include `<FOO>_NO_OPTIMISE = -O0` flags so that Cactus can explicitly disable optimization as needed for compilers that optimize by default (Intel and Cray).

This finishes compiler options and options for ExtenalLibraries are next.

For `BLAS` / `LAPACK` and the Intel compiler, I want to use Intel's `MKL` library which is included with the compiler. As such there is a shorcut use it by specifying `-mkl` to the compiler. Usually in the simfactory option lists I do this by (ab-)using `BLAS_LIBS` and `LAPACK_LIBS` :

```
BLAS_DIR  = NO_BUILD
BLAS_LIBS = -mkl

LAPACK_DIR  = NO_BUILD
LAPACK_LIBS = -mkl
```

which occasionally requires that one adds the path to the directory containing `MKL` libraries to the `--rpath` option of the linker so that the library files are found at runtime. This is done by setting `LDFLAGS = -W,--rpath,<PATH-TO-MKL>` where the correct path is a bit of guesswork. The output of `module show intel/19.1.1` is very useful for this sine it list the compiler installation directory:

```
> module show intel/19.1.1
----------------------------------------------------------------------------
------------------------
   /glade/u/apps/ch/modulefiles/default/compilers/intel/19.1.1.lua:
----------------------------------------------------------------------------
------------------------
family("compiler")
whatis("intel v19.1.1")
help([[This module loads the Intel Compilers:
  C:        icc
  C++:      icpc
  Fortran:  ifort
For more information on the individual compilers and their suboptions
refer to the man page for the individual compilers.

Note that compiler version 19.1.1 is part of version 2020u1
of the Intel Parallel Studio.

Website: https://software.intel.com/en-us/parallel-studio-xe
]])
prepend_path("PATH","/glade/u/apps/opt/intel/2020u1/compilers_and_librarie
s/linux/bin/intel64")
prepend_path("LD_LIBRARY_PATH","/glade/u/apps/opt/intel/2020u1/compilers_an
d_libraries/linux/lib/intel64")
```

where `PATH` and `LD_LIBRARY_PATH` are of interest. Sometimes `MLROOT` or `MKLHOME` may also be set which are best. In any case we are looking for paths that end in `mkl/lib/intel64` (for `MKL` ) and `linux/lib/intel64` (for generic runtime libs).

At this point I also look for other modules that I can use to satisfy ExternalLibraries. For example `fftw/3.3.8` for `FFTW3` . Similar to the `MKL` example `module show fftw/3.3.8` provides clues for what paths to use and libraries to link against. While this is not guaranteed it works quite often.

Each module used this way should be added to the `envsetup` option of the machine ini file (with full path) like so:

```
module load fftw/3.3.8 &&
```

where the `&&` at the end serves to propagate module load errors to simfactory so that it aborts a build if a module fails to load (more on this when setting up the ini file).

Some ExternalLibraries offer a `<FOO>_EXTRA_LIBS` variable to specify extra depenencies to link against. This is often required for `HWLOC` which can detect and use `libnuma` for non-uniform memory layouts but detects it in such a manner that Cactus cannot automatically handle it. Thus often

```
HWLOC_EXTRA_LIBS = numa
```

is required to avoid strange link time errors about missing symbols (mostly for static lining).

Another often useful trick is to set `<FOO>_DIR = BUILD` which forces the ExternalLibrary to be built by Cactus using the included tarball.

**Option list changes**

This time the changes are quite extensive:

```
--- simfactory/mdb/optionlists/shelob-openmpi.cfg       2021-08-31 09:57:0
9.537577000 -0600
+++ simfactory/mdb/optionlists/cheyenne.cfg     2021-09-16 12:51:19.1806930
00 -0600
@@ -1,24 +1,22 @@
 # Whenever this version string changes, the application is configured
 # and rebuilt from scratch
-VERSION = shelob-openmpi-2014-10-21
+VERSION = cheyenne-2021-09-01

 CPP = cpp
 FPP = cpp
-CC  = /usr/local/compilers/Intel/composer_xe_2013.5.192/bin/intel64/icc
-CXX = /usr/local/compilers/Intel/composer_xe_2013.5.192/bin/intel64/icpc
-F77 = /usr/local/compilers/Intel/composer_xe_2013.5.192/bin/intel64/ifort
-F90 = /usr/local/compilers/Intel/composer_xe_2013.5.192/bin/intel64/ifort
-
-CPPFLAGS = -DMPICH_IGNORE_CXX_SEEK
-FPPFLAGS = -traditional
-CFLAGS   = -g -xHOST -align -std=gnu99 -U__STRICT_ANSI__ -ansi_alias
-CXXFLAGS = -g -xHOST -align -std=gnu++11 -U__STRICT_ANSI__ -ansi_alias
+CC  = icc
+CXX = icpc
+F90 = ifort
+
+FPPFLAGS = --traditional
+CFLAGS   = -g -xHOST -align -std=gnu99
+CXXFLAGS = -g -xHOST -align -std=gnu++11
 F77FLAGS = -g -xHOST -align -pad -safe_cray_ptr
 F90FLAGS = -g -xHOST -align -pad -safe_cray_ptr

-LDFLAGS = -Wl,-rpath,/usr/local/compilers/Intel/composer_xe_2013.5.192/com
piler/lib/intel64 -Wl,-rpath,/usr/local/compilers/Intel/composer_xe_2013.5.
192/mkl/lib/intel64
-LIBDIRS = /usr/local/packages/cuda/7.5/lib64   /usr/local/compilers/Intel/
composer_xe_2013.5.192/compiler/lib/intel64 /usr/local/compilers/Intel/comp
oser_xe_2013.5.192/mkl/lib/intel64
-LIBS    = cudart   ifcore imf svml numa
+LDFLAGS = -Wl,-rpath,/glade/u/apps/opt/intel/2019u5/compilers_and_librarie
s/linux/lib/intel64 -Wl,-rpath,/glade/u/apps/opt/intel/2019u5/compilers_and
_libraries/linux/mkl/lib/intel64
+#LIBDIRS = /usr/local/packages/cuda/7.5/lib64   /usr/local/compilers/Inte
l/composer_xe_2013.5.192/compiler/lib/intel64 /usr/local/compilers/Intel/co
mposer_xe_2013.5.192/mkl/lib/intel64
+LIBS    = ifcore

 C_LINE_DIRECTIVES = yes
 F_LINE_DIRECTIVES = yes
@@ -27,7 +25,6 @@
 VECTORISE_ALIGNED_ARRAYS = no
 VECTORISE_INLINE         = no
```

```
-# -check-uninit fails for asm output operands
 DEBUG           = no
 CPP_DEBUG_FLAGS = -DCARPET_DEBUG
 FPP_DEBUG_FLAGS = -DCARPET_DEBUG
@@ -75,58 +72,33 @@
 F77_WARN_FLAGS =
 F90_WARN_FLAGS =

-CUDA_DIR         = /usr/local/packages/cuda/7.5
-CUCC             = /usr/local/packages/cuda/7.5/bin/nvcc
-CUCC_EXTRA_FLAGS = --maxrregcount 60 -Drestrict=__restrict__
-
-

 BLAS_DIR  = NO_BUILD
 BLAS_LIBS = -mkl

-# CUDA, configured with the flesh
-CUCC                = /usr/local/packages/cuda/7.5/bin/nvcc
-CUCCFLAGS           = -m 64 -arch sm_35 -g -maxrregcount 60 -Drestrict=__r
estrict__
-CUCC_OPTIMISE_FLAGS = -O3 -use_fast_math
-CUCC_WARN_FLAGS     =
+FFTW3_DIR = /glade/u/apps/ch/opt/fftw/3.3.8/intel/19.0.5

-# CUDA, configured via thorn CUDA
-CUDA_DIR                    = /usr/local/packages/cuda/7.5
-CUCC_ARCH                   = sm_35
-CUCC_EXTRA_FLAGS            = -maxrregcount 63 -Drestrict=__restrict__
-CUCC_EXTRA_OPTIMISE_FLAGS = -use_fast_math
-CUCC_EXTRA_WARN_FLAGS     = -use_fast_math
-
-# FFTW3_DIR = /usr/local/packages/fftw/3.3.3/Intel-13.0.0
-FFTW3_DIR = BUILD
+GSL_DIR = /glade/u/apps/ch/opt/gsl/2.6/intel/19.0.5

-GSL_DIR = /usr/local/packages/gsl/1.15/Intel-13.0.0
+HDF5_DIR = /glade/u/apps/ch/opt/netcdf/4.8.0/intel/19.0.5

-HDF5_DIR = /usr/local/packages/hdf5/1.8.10/Intel-13.0.0-openmpi-1.6.2
-# HDF5_DIR = BUILD
-
-HWLOC_EXTRA_LIBS = numa
+#HWLOC_EXTRA_LIBS = numa

 LAPACK_DIR  = NO_BUILD
 LAPACK_LIBS = -mkl

-MPI_DIR      = /usr/local/packages/openmpi/1.6.5/Intel-13.0.0
-MPI_INC_DIRS = /usr/local/packages/openmpi/1.6.5/Intel-13.0.0/include
-MPI_LIB_DIRS = /usr/local/packages/openmpi/1.6.5/Intel-13.0.0/lib /usr/lib
64
```

```
-MPI_LIBS     = mpi open-rte open-pal rdmacm ibverbs ibumad util
+MPI_DIR      = /glade/u/apps/ch/opt/openmpi/4.0.5/intel/19.0.5
+MPI_LIBS    =  mpi mpi_usempif08

 OPENBLAS_DIR  = NO_BUILD
 OPENBLAS_LIBS = -mkl

-OPENCL_DIR = /usr/local/packages/cuda/7.5
-
-PAPI_DIR  = /home/diener/papi-5.1.0.2-intel
-PAPI_LIBS = papi
+PAPI_DIR  = BUILD

 PETSC_DIR                     = BUILD
-PETSC_BLAS_EXTRA_LIB_DIRS   = /usr/local/compilers/Intel/composer_xe_2013.
5.192/mkl/lib/intel64
+PETSC_BLAS_EXTRA_LIB_DIRS   = /glade/u/apps/opt/intel/2019u5/compilers_and
_libraries/linux/mkl/lib/intel64
 PETSC_BLAS_EXTRA_LIBS       = mkl_intel_lp64 mkl_intel_thread mkl_core iom
p5
-PETSC_LAPACK_EXTRA_LIB_DIRS = /usr/local/compilers/Intel/composer_xe_2013.
5.192/mkl/lib/intel64
+PETSC_LAPACK_EXTRA_LIB_DIRS = /glade/u/apps/opt/intel/2019u5/compilers_and
_libraries/linux/mkl/lib/intel64
 PETSC_LAPACK_EXTRA_LIBS     = mkl_intel_lp64 mkl_intel_thread mkl_core iom
p5

 PTHREADS_DIR = NO_BUILD
```

## Machine ini file

The ini file ties al information together by providing information direclty to simfactory. It has sufficient information for simfactory to identify the machine and indicates which option list, submit script and run script to use.

Since it interacts with the resource manager, it usually best to use the same machine as "donor" for both the submitscript and machine ini templates. In my case, this is BlueWaters:

```
> cp simfactory/mdb/machines/bluewaters.ini simfactory/mdb/machines/cheyenn
e.ini
```

Editing the file it is important to adjust the section name since this is the name that simfactory will use for the machine. So I change `[bluewaters]` to `[cheyenne]` .

Using the correct template means that many tricky fields, like `submitpattern` are already corect or at least filled with easily fixable guesses.

Similarly you should update the `envsetup` setting to load all modules that were used to provide ExternalLibraries in the option list `cheyenne.cfg` (thus you may want to have both files open and edit them incrementally). The shell command in `envsetup` is execute before simfactory builds, submits or otherwise runs shell commands and loads required modules. Simfactory aborts if this command fails.

It is usually best to link all lines with `&&` to propagate module load error to simfactory so that it can abort the build process. Personally I always list the exact versions required and also add `module unload <foo>` lines at the beginning for all modules I will later load to avoid module load conflicts. You may also have to add a line like `source /etc/profile` at the very start to ensure that the `module command` itself is available by (re)executing the shell's start files (profile) in the shell that simfactory starts.

Most option are fairly straightforward to set by consulting the machine documentation pages, e.g. https://arc.ucar.edu/knowledge_base/70549542 (https://arc.ucar.edu/knowledge_base/70549542) for homepage, machine name, login hostname to use, flop counts etc. (though WikiPedia can also be a good source for this "information only" fields).

They are all extensively documented in the ET wiki's (https://docs.einsteintoolkit.org/et-docs/Configuring_a_new_machine) "Configuring a new machine" page.

`aliaspattern` should be set to a regular expression that matches the login node hostnames for this machine (and only this machine). If you did not note it down, `./simfactory/bin/sim whoami` will show it again (until you have created the ini file). For CHeyenne a good guess is:

```
aliaspattern = ^cheyenne[1-9]\.cheyenne\.ucar\.edu$
```

where the anchors `^` and `$` ensure that the full hostname is considered.

Once you are happy the ini file you can test `envsetup` and the `aliaspattern` using:

```
> ./simfactory/bin/sim whoami
> ./simfactory/bin/sim execute 'echo "Hello, world!"'
```

which should output the detected machine name (cheyenne) in the first line. The second line instructs simfactory to run the systems `echo` command to print "Hello, world!" using the `envsetup` command.

On Cheyenne I used this to fix module conflicts between `HDF5` (which I want) and `netcdf` (loaded by default but I don't care).

#### Machine ini diff

```
> diff -uw simfactory/mdb/machines/bluewaters.ini simfactory/mdb/machines/c
heyenne.ini
--- simfactory/mdb/machines/bluewaters.ini      2021-08-31 09:57:09.4562620
00 -0600
+++ simfactory/mdb/machines/cheyenne.ini        2021-09-16 13:10:28.2465750
00 -0600
@@ -1,91 +1,74 @@
-[bluewaters]
+[cheyenne]

-# last-tested-on: 2016-06-04
-# last-tested-by: Erik Schnetter <schnetter@gmail.com>
+# last-tested-on:
+# last-tested-by: Roland Haas <rhaas@illinois.edu>

 # Machine description
-nickname        = bluewaters
-name            = Blue Waters
-location        = NCSA
-description     = A Cray XE6/XK7 at the NCSA
-webpage         = https://bluewaters.ncsa.illinois.edu/user-guide
+nickname        = cheyenne
+name            = Cheynne
+location        = NCAR / UCAR
+description     = An SGI ICE XA Cluster
+webpage         = https://www2.cisl.ucar.edu/resources/computational-syste
ms/cheyenne
 status          = production

 # Access to this machine
-hostname        = h2ologin-duo.ncsa.illinois.edu
+hostname        = cheyenne.ucar.edu
 sshcmd          = ssh
 sshopts         =

-# CUDA disabled b/c CUDA requires gcc 4.9 instead of 6.3
 envsetup        = <<EOT
     source /etc/profile
-    module unload PrgEnv-cray PrgEnv-gnu PrgEnv-intel PrgEnv-pathscale Prg
Env-pgi
-    module load PrgEnv-gnu/5.2.82
-    module unload gcc
-    module unload OpenSSL
-    module load gcc/6.3.0
-    module load atp
-    module load boost/1.63.0
-    module load cray-hdf5-parallel/1.10.2.0
-    module load cray-petsc/3.9.3.0
-    module load cray-fftw/3.3.6.5
-    module load gsl/1.16
-    module load make/3.82
-    module load papi/5.5.0.1
-    module load OpenSSL/1.0.2m
-    module load pmi
+    module unload intel openmpi netcdf mpt ncarcompilers &&
+    module load intel/19.0.5 &&
+    module load ncarcompilers/0.5.0 &&
+    module load openmpi/4.0.5 &&
+    module load gsl/2.6 &&
+    module load fftw/3.3.8 &&
+    module load hdf5/1.10.7 &&
+    unset HDF5
 EOT
-aliaspattern    = ^h2ologin[1-4](\.ncsa\.illinois\.edu)?$
+aliaspattern    = ^cheyenne[1-9].cheyenne\.ucar\.edu$

 # Source tree management
-sourcebasedir   = /u/sciteam/@USER@
+sourcebasedir   = /glade/u/home/@USER@
 disabled-thorns = <<EOT
 EOT
 enabled-thorns = <<EOT
     ExternalLibraries/pciutils
 EOT
-# GPU using thorns
-#   CactusExamples/HelloWorldCUDA
-#   ExternalLibraries/OpenCL
-#   CactusExamples/HelloWorldOpenCL
-#   CactusExamples/WaveToyOpenCL
-#   CactusUtils/OpenCLRunTime
-#   CactusUtils/Accelerator
-#   McLachlan/ML_BSSN_CL
-#   McLachlan/ML_BSSN_CL_Helper
-#   McLachlan/ML_WaveToy_CL
-optionlist      = bluewaters-gnu.cfg
-submitscript    = bluewaters.sub
-runscript       = bluewaters.run
-makejobs        = 16
+optionlist      = cheyenne.cfg
+submitscript    = cheyenne.sub
+runscript       = cheyenne.run
+makejobs        = 4
 make            = make -j@MAKEJOBS@

 # Simulation management
-basedir         = /scratch/sciteam/@USER@/simulations
-cpu             = AMD Opteron(TM) Processor 6276
-cpufreq         = 2.45   # 2.3 is guaranteed, 2.45 is average
-flop/cycle      = 4
+basedir         = /glade/scratch/@USER@/simulations
+cpu             = 2.3-GHz Intel Xeon E5-2697V4 (Broadwell) processors 16 f
lops per clock
+cpufreq         = 2.3
+flop/cycle      = 16
```

```
 max-num-smt     = 1
 num-smt         = 1
-ppn             = 32
-mpn             = 4
-max-num-threads = 32
-num-threads     = 8
+ppn             = 36
+#mpn             = 4
+max-num-threads = 36
+num-threads     = 6
 memory          = 65536
-nodes           = 25712
+nodes           = 4032
 min-ppn         = 1
 allocation      = NO_ALLOCATION
-queue           = normal:xe     # or normal:xk
-maxwalltime     = 48:00:00
+# see https://www2.cisl.ucar.edu/resources/computational-systems/cheyenne/
running-jobs/job-submission-queues
+# for available queue: premium, regular, economy, share
+queue           = regular
+maxwalltime     = 12:00:00
 submit          = cd @RUNDIR@ && qsub @SCRIPTFILE@
 getstatus       = qstat @JOB_ID@
 stop            = qdel @JOB_ID@
-submitpattern   = (\d+[.]bw)
+submitpattern   = (\d+)\.chadmin1
 statuspattern   = "^@JOB_ID@[. ].* [QRH] "
 queuedpattern   = "^@JOB_ID@[. ].* Q "
 runningpattern  = "^@JOB_ID@[. ].* R "
 holdingpattern  = "^@JOB_ID@[. ].* H "
-#scratchbasedir   = /lustre/scratch/@USER@
 exechost        = hostname
 exechostpattern = (.*)
 stdout          = cat @SIMULATION_NAME@.out
```

## Compiling for the first ime

With this in place you can now try to compile using:

```
> ./simfactory/bin/sim build --thornlist manifest/einsteintoolkit.th etk
```

In may case this resulted in a CST time failure:

```
Don't understand the setting "HDF/glade/u/apps/ch/opt/netcdf/4.8.0/intel/1
9.0.5/" !
```

which due to some of the NCAR modules setting an environment variable `HDF5` which conflicts (and overrides) any option list variable `HDF5`.

In my case the solution is to add `undef HDF5` ot `envsetup`.

## Updating option lists and recompiling

You must use the `--reconfig` option of simfactory *and* explicitly give the new (even if same file name) option list as an `--optionlist simfactory/mdb/optionlists/cheyenne.cfg` to force an update. Using `make etk-cleandeps` may also be useful.

Similarly for `--runscript` and `--submitscript`, but not for the ini file.

In [ ]:

```

```