WDMapp

Oct 09, 2020

Contents:

CHAPTER 1

Overview

The Whole Device Model Application (WDMApp) in the DOE Exascale Computing Project (ECP) is developing a high-fidelity model of magnetically confined fusion plasmas, urgently needed to plan experiments on ITER and optimize the design of next-step fusion facilities. These devices will operate in high-fusion-gain physics regimes not achieved by any experiment, making predictive numerical simulation the best tool for the task. WDMApp is focused on building the main driver and coupling framework for a WDM. The main driver is based on the coupling of two advanced and highly scalable gyrokinetic codes, XGC and GENE, where the former is a particle-in-cell code optimized for the treating the edge plasma while the other is a continuum code optimized for the core. As an alternative, the GEM gyrokinetic code can be used instead of GENE to simulate the core region. WDMApp aims to take advantage of the complementary nature of the simulation codes to build the most advanced and efficient whole device kinetic transport kernel for the WDM. A major part of the technical development work is targeting the coupling framework, which will be further developed for exascale and optimized for coupling most of the physics modules that will operate at various space and time scales. The current MPI +X is to be enhanced with communication-avoiding methods, task-based parallelism, in situ analysis with resources for load optimization workflows, and deep memory hierarchy-aware algorithms.

Sample simulation results from the WDMApp codes are available at http://wdmapp.pppl.gov/.

CHAPTER 2

Building and Running WDMapp

In the following, we provide instructions on how to build and run WDMapp on specific machines. We are using the Spack package manager, so it should be relatively straightforward to install on other machines as well, see also the generic instructions below.

Overall, the process works like this:

- Apply for access
- Install Spack and customize for your machine.
- Add the WDMapp Spack repo.
- Build and install WDMapp.
- Provide input parameters and setup for simulation run.
- Submit a job.

2.1 Applying for Access

2.1.1 Overview

The Whole Device Model Application (WDMApp) uses access controlled versions of XGC, GENE, GEM and supporting tools. Before installing these packages users must first apply for access. Specific instructions for each package, and WDMApp are given below.

2.1.2 XGC

Visit https://www.pppl.gov/organization/technology-transfer/software-access-request for instructions. Be sure to reference the WDMApp ECP project and list *XGC-Develop* as the needed software.

2.1.3 GENE

Visit http://genecode.org/ then click the 'Get GENE' link at the top and fill out the form.

2.1.4 GEM

Visit http://www.gemgyrokinetic.org/, click the 'Download File' link at the bottom and follow the instructions within the form.

2.1.5 WDMAPP

Once you have submitted applications to the above, and access has been granted, please send your GitHub and GENE GitLab usernames to *kai dot germaschewski at unh dot edu* and *smithc11 at rpi dot edu* so we can verify your access to XGC, GENE, and GEM and give you membership to the WDMApp GitHub Organization.

2.2 WDMApp on Summit at OLCF

2.2.1 Setting up Spack

Spack is a generic package manager for HPC. We rely on it in the following to install WDMapp and its components. Setting up Spack is a one-time process on a given machine – if you already have a working Spack install, you should be able to use it. However, in practice there are plenty of ways that things can wrong, so we provide tested Spack setups for a selection of machines. Following our instructions makes sure that WDMapp is built with compatible compilers and machine-specific system packages (e.g., MPI, CUDA, etc.).

2.2.2 Installing Spack

Follow the instructions from the Spack Documentation.

```
$ git clone -b v0.15.4 https://github.com/spack/spack.git
```

Note: v0.15.4 is the latest spack stable version on 2020-10-20; newer versions will likely work but have not been tested. Using the default 'develop' branch is not recommended, as it does break sometimes and introduces a lot of package version churn if you try to track it.

Enable shell support for Spack.

```
# For bash/zsh users
$ export SPACK_ROOT=/path/to/spack
$ . $SPACK_ROOT/share/spack/setup-env.sh

# For tcsh or csh users (note you must set SPACK_ROOT)
$ setenv SPACK_ROOT /path/to/spack
$ source $SPACK_ROOT/share/spack/setup-env.csh
```

2.2.3 Cloning the WDMapp package repo

Just clone the repository from github to the same machine that you just set up Spack on.

```
$ git clone git@github.com:wdmapp/wdmapp-config.git
```

Summit-Specific Setup

Rhea and Summit share a common home directory. If you use Spack on both machines, this leads to issues because both instances will share their config files, which by default go into `~/.spack/linux`. If you only want to use just one or the other machine, you can ignore the following note.

Note: One way to deal with keeping separate spack setups on Rhea and Summit is to make separate `~/.spack/linux-rhea` and `~/spack/linux-summit` directories and symlink one or the other to `~/.spack/linux`

```
$ # make sure ~/.spack/linux does exit yet -- if it does, move it  out of the way
$ mkdir -p ~/.spack/linux-rhea
$ mkdir -p ~/.spack/linux-summit
$ ln -snf ~/.spack/linux-rhea ~/.spack/linux # if on rhea
$ ln -snf ~/.spack/linux-summit ~/.spack/linux # if on summit
```

An alternative is to have two separate spack installs, and instead of keeping the config files in `~/.spack`, they can be put into `$SPACK_ROOT/etc/spack`, so with two different roots they can be kept separate. You can then do this in your `.bashrc`:

```
if [ `uname -m` == "ppc64le" ]; then
  export SPACK_ROOT=$HOME/spack-summit
else
  export SPACK_ROOT=$HOME/spack-rhea
fi
source $SPACK_ROOT/share/spack/setup-env.sh
```

Spack commands that edit configuration files such as `spack compiler add` can be called `spack compiler add --scope site` to update files living in `$SPACK_ROOT/etc/spack`.

Employing our provided Spack configuration

Warning: The folllowing will overwrite an existing Spack configuration, so be careful if you've previously set up Spack. If you have an existing config, consider renaming `~./spack` to back it up.

Just copy the provided YAML configuration files to where Spack expects them:

```
$ mkdir -p ~/.spack/linux
$ cp path/to/wdmapp-config/summit/spack/*.yaml ~/.spack/linux
```

You can have a choice of a basic or a more comprehensive setup for Spack on Summit from the wdmapp-config repository.

If you use the provided `packages.yaml`, it only tells Spack about essential existing pre-installed packages on Summit, ie., CUDA, MPI and the corresponding compilers. Spack will therefore build and install all other dependencies

from scratch, which takes time but has the advantage that it'll generate pretty much the same software stack on any machine you use.

On the other hand, `packages-extended.yaml` (which needs to be renamed to `packages.yaml` to be used), tells Spack comprehensively about pre-installed software on Summit, so installation of WDMapp will proceed more quickly and use system-provided libraries where possible.

Warning: Make sure that you don't have `xl` or `spectrum-mpi` loaded. By default, Summit will load the `xl` and `spectrum-mpi` modules for you, and those interfere when Spack tries to perform `gcc` based builds. You might want to consider adding this to your `.bashrc` or similar init file:

```
module unload xl spectrum-mpi
```

Note: On Summit, the cuda module sets environment variables that set a path which `nvcc` does not otherwise add. Because of this, it is requried to `module load cuda/10.1.243` before building GENE, and probably other software that uses CUDA..

Note: Consider also configuring spack to use gpfs scratch space (i.e. `$MEMBERWORK`) as an alternative when building packages, in addition to the default `tmpfs` and home filesystem (which can have problems with high workload tasks):

```
$ mkdir -p /gpfs/alpine/scratch/$USER/<project-id>/spack-stage
```

and add the following to `~/.spack/config.yaml` (`$SPACK_ROOT/etc/spack/config.yaml`). Spack tries each entry in order for precedence:

```
config:
  build_stage:
    - $tempdir/$user/spack-stage
    - /gpfs/alpine/scratch/$USER/<project-id>/spack-stage
    - ~/.spack/stage
```

2.2.4 Adding the WDMapp package repo to Spack

This will let Spack search the WDMapp repository for packages that aren't found in its builtin package repository.

```
$ spack repo add path/to/wdmapp-config/spack/wdmapp
==> Added repo with namespace 'wdmapp'.
```

Note: To update the wdmapp package repository to the latest, just run `git pull` in the directory where you cloned `wdmapp-config/`.

2.2.5 Building WDMapp

You should be able to just follow the generic instructions from *Building WDMAPP*.

2.2.6 Running the Cyclone Test Case

Enable shell support for Spack:

```
# For bash/zsh users
$ export SPACK_ROOT=/path/to/spack
$ . $SPACK_ROOT/share/spack/setup-env.sh

# For tcsh or csh users (note you must set SPACK_ROOT)
$ setenv SPACK_ROOT /path/to/spack
$ source $SPACK_ROOT/share/spack/setup-env.csh
```

Load the wdmapp modules:

```
$ spack load wdmapp arch=linux-rhel7-power9le
$ spack load effis +compose arch=linux-rhel7-power9le
```

Clone the testcases repo:

```
$ git clone git@github.com:wdmapp/testcases.git
$ cd testcases/run_1/summit
```

See the *Workflow Composition in EFFIS* page for help editing the workflow composition file. As quick pointers, make sure to edit the path to the run directory (`/path/to/testDir` below) called `rundir`, the binaries labeled `executable_path`, and the project, `charge`, in `run_1.yaml`.

Since we loaded the `wdmapp` module via Spack the binaries are in your `PATH` and their location can be retrieved with the `which xgc-es gene cpl` command.

Run the effis pre-processor:

```
$ effis-compose.py run_1.yaml
```

Submit the job:

```
$ effis-submit /path/to/testDir
```

2.3 WDMApp on Rhea at OLCF

2.3.1 Setting up Spack

Spack is a generic package manager for HPC. We rely on it in the following to install WDMapp and its components. Setting up Spack is a one-time process on a given machine – if you already have a working Spack install, you should be able to use it. However, in practice there are plenty of ways that things can wrong, so we provide tested Spack setups for a selection of machines. Following our instructions makes sure that WDMapp is built with compatible compilers and machine-specific system packages (e.g., MPI, CUDA, etc.).

2.3.2 Installing Spack

Follow the instructions from the Spack Documentation.

```
$ git clone -b v0.15.4 https://github.com/spack/spack.git
```

Note: v0.15.4 is the latest spack stable version on 2020-10-20; newer versions will likely work but have not been tested. Using the default 'develop' branch is not recommended, as it does break sometimes and introduces a lot of package version churn if you try to track it.

Enable shell support for Spack.

```
# For bash/zsh users
$ export SPACK_ROOT=/path/to/spack
$ . $SPACK_ROOT/share/spack/setup-env.sh

# For tcsh or csh users (note you must set SPACK_ROOT)
$ setenv SPACK_ROOT /path/to/spack
$ source $SPACK_ROOT/share/spack/setup-env.csh
```

2.3.3 Cloning the WDMapp package repo

Just clone the repository from github to the same machine that you just set up Spack on.

```
$ git clone git@github.com:wdmapp/wdmapp-config.git
```

2.3.4 Rhea-Specific Setup

Rhea and Summit share a common home directory. If you use Spack on both machines, this leads to issues because both instances will share their config files, which by default go into `~/.spack/linux`. If you only want to use just one or the other machine, you can ignore the following note.

Note: One way to deal with keeping separate spack setups on Rhea and Summit is to make separate `~/.spack/linux-rhea` and `~/spack/linux-summit` directories and symlink one or the other to `~/.spack/linux`

```
$ # make sure ~/.spack/linux does exit yet -- if it does, move it  out of the way
$ mkdir -p ~/.spack/linux-rhea
$ mkdir -p ~/.spack/linux-summit
$ ln -snf ~/.spack/linux-rhea ~/.spack/linux # if on rhea
$ ln -snf ~/.spack/linux-summit ~/.spack/linux # if on summit
```

An alternative is to have two separate spack installs, and instead of keeping the config files in `~/.spack`, they can be put into `$SPACK_ROOT/etc/spack`, so with two different roots they can be kept separate. You can then do this in your `.bashrc`:

```
if [ `uname -m` == "ppc64le" ]; then
  export SPACK_ROOT=$HOME/spack-summit
else
  export SPACK_ROOT=$HOME/spack-rhea
fi
source $SPACK_ROOT/share/spack/setup-env.sh
```

Spack commands that edit configuration files such as `spack compiler add` can be called `spack compiler add --scope site` to update files living in `$SPACK_ROOT/etc/spack`.

Employing our provided Spack configuration

Warning: The folllowing will overwrite an existing Spack configuration, so be careful if you've previously set up Spack. If you have an existing config, consider renaming `~./spack` to back it up.

Just copy the provided YAML configuration files to where Spack expects them:

```
$ mkdir -p ~/.spack/linux
$ cp path/to/wdmapp-config/rhea/spack/*.yaml ~/.spack/linux
```

On Rhea an `olcf` repo is also needed to make it possible to use system-installed packages from our Spack. This repo is provided by the *wdmapp-config* you cloned earlier:

```
$ spack repo add path/to/wdmapp-config/rhea/spack/olcf
==> Added repo with namespace 'olcf'
```

Note: Consider also configuring spack to use gpfs scratch space (i.e. `$MEMBERWORK`) as an alternative when building packages, in addition to the default `tmpfs` and home filesystem (which can have problems with high workload tasks):

```
$ mkdir -p /gpfs/alpine/scratch/$USER/<project-id>/spack-stage
```

and add the following to `~/.spack/config.yaml` (`$SPACK_ROOT/etc/spack/config.yaml`). Spack tries each entry in order for precedence:

```
config:
  build_stage:
    - $tempdir/$user/spack-stage
    - /gpfs/alpine/scratch/$USER/<project-id>/spack-stage
    - ~/.spack/stage
```

2.3.5 Adding the WDMapp package repo to Spack

This will let Spack search the WDMapp repository for packages that aren't found in its builtin package repository.

```
$ spack repo add path/to/wdmapp-config/spack/wdmapp
==> Added repo with namespace 'wdmapp'.
```

Note: To update the wdmapp package repository to the latest, just run `git pull` in the directory where you cloned `wdmapp-config/`.

Note: The E4S project has created a build cache for Rhea. This provides many packages as precompiled binaries, so will reduce the installation time. To use it:

```
$ wget https://oaciss.uoregon.edu/e4s/e4s.pub
$ spack gpg trust e4s.pub
$ spack mirror add E4S https://cache.e4s.io
```

2.3.6 Building WDMapp

You should be able to just follow the generic instructions from *Building WDMAPP*.

Using E4S WDMapp docker container

Alternatively, the E4S project has created a docker image that mirrors the Rhea environment, which can be used for local development and debugging. To run this image, you need to have docker installed and then do the following:

```
$ docker pull ecpe4s/ubi7.7_x86_64_base_wdm:1.0
$ docker run --rm -it ecpe4s/ubi7.7_x86_64_base_wdm:1.0
```

In order for the image to get the access controlled components, you need to provide it with your private SSH key that provides access to the respective private github repos. In the image, do the following in the docker image:

```
# cat > .ssh/id_rsa # Then copy&paste your private key
# chmod 600 .ssh/id_rsa
```

This provides an development environment with everything but the private codes preinstalled. All that's needed to complete building and installing them is:

```
# spack install wdmapp target=x86_64
```

2.3.7 Running the Cyclone Test Case

Enable shell support for Spack:

```
# For bash/zsh users
$ export SPACK_ROOT=/path/to/spack
$ . $SPACK_ROOT/share/spack/setup-env.sh

# For tcsh or csh users (note you must set SPACK_ROOT)
$ setenv SPACK_ROOT /path/to/spack
$ source $SPACK_ROOT/share/spack/setup-env.csh
```

Load the wdmapp modules:

```
$ spack load wdmapp arch=linux-rhel7-sandybridge
$ spack load effis +compose arch=linux-rhel7-sandybridge
```

Clone the testcases repo:

```
$ git clone git@github.com:wdmapp/testcases.git
$ cd testcases/run_1/rhea
```

See the *Workflow Composition in EFFIS* page for help editing the workflow composition file. As quick pointers, make sure to edit the path to the run directory (`/path/to/testDir` below) called `rundir`, the binaries labeled `executable_path`, and the project, `charge`, in `run_1.yaml` (or `run_externalCpl.yaml` if *wdmapp+passthrough* was built in *Building WDMAPP*).

Since we loaded the `wdmapp` module via Spack the binaries are in your `PATH` and their location can be retrieved with the `which xgc-es gene cpl` command.

Run the effis pre-processor:

```
$ effis-compose.py run_1.yaml
```

Submit the job:

```
$ effis-submit /path/to/testDir
```

2.3.8 Running the Cyclone Test Case - External Coupler

The cyclone test case can be executed with the external coupler (`wdmapp+passthrough` built in *Building WDMAPP* by following the instructions for *Running the Cyclone Test Case* using `run_externalCpl.yaml` instead of `run_1.yaml`.

2.4 WDMapp on Longhorn at TACC

2.4.1 Setting up Spack

Spack is a generic package manager for HPC. We rely on it in the following to install WDMapp and its components. Setting up Spack is a one-time process on a given machine – if you already have a working Spack install, you should be able to use it. However, in practice there are plenty of ways that things can wrong, so we provide tested Spack setups for a selection of machines. Following our instructions makes sure that WDMapp is built with compatible compilers and machine-specific system packages (e.g., MPI, CUDA, etc.).

2.4.2 Installing Spack

Follow the instructions from the Spack Documentation.

```
$ git clone -b v0.15.4 https://github.com/spack/spack.git
```

Note: v0.15.4 is the latest spack stable version on 2020-10-20; newer versions will likely work but have not been tested. Using the default 'develop' branch is not recommended, as it does break sometimes and introduces a lot of package version churn if you try to track it.

Enable shell support for Spack.

```
# For bash/zsh users
$ export SPACK_ROOT=/path/to/spack
$ . $SPACK_ROOT/share/spack/setup-env.sh

# For tcsh or csh users (note you must set SPACK_ROOT)
$ setenv SPACK_ROOT /path/to/spack
$ source $SPACK_ROOT/share/spack/setup-env.csh
```

2.4.3 Cloning the WDMapp package repo

Just clone the repository from github to the same machine that you just set up Spack on.

```
$ git clone git@github.com:wdmapp/wdmapp-config.git
```

2.4.4 Longhorn-Specific Setup

Employing our provided Spack configuration

Warning: The folllowing will overwrite an existing Spack configuration, so be careful if you've previously set up Spack. If you have an existing config, consider renaming `~./spack` to back it up.

Just copy the provided YAML configuration files to where Spack expects them:

```
$ mkdir -p ~/.spack/linux
$ cp path/to/wdmapp-config/longhorn/spack/*.yaml ~/.spack/linux
```

Note: Make sure that you don't have a `spectrum-mpi` loaded. By default, Longhorn will load the `xl` and `spectrum-mpi` modules for you, and those interfere when Spack tries to perform `gcc` based builds. You might want to consider adding this to your `.bashrc` or similar init file:

```
module unload xl spectrum-mpi
```

Creating your own setup for Longhorn

Alternatively, you can create your own config:

```
module load gcc/7.3.0
spack compiler find
```

This should find a number of compilers, including gcc 7.3.0. You may want to repeat this step for gcc 9.1.0 – however, there is currently no preinstalled MPI for this compiler.

The compilers on longhorn require `LD_LIBRARY_PATH` hackery to function, and spack sanitizes `LD_LIBRARY_PATH`. The workaround is described here. In this case, edit `~/.spack/linux/compilers.yaml` using `spack config edit compilers` and modify the modules section for gcc 7 like this:

```
- compiler:
   spec: gcc@7.3.0
   paths:
     cc: /opt/apps/gcc/7.3.0/bin/gcc
     cxx: /opt/apps/gcc/7.3.0/bin/g++
     f77: /opt/apps/gcc/7.3.0/bin/gfortran
     fc: /opt/apps/gcc/7.3.0/bin/gfortran
   flags: {}
   operating_system: rhel7
   target: ppc64le
   modules: [gcc/7.3.0] # <-- ADD THIS
   environment: {}
   extra_rpaths: []

 similar is required for gcc/9.1.0, and possibly for xl.
```

Add a minimal `packages.yaml` in `~/.spack/linux/packages.yaml` that registers the preinstalled openmpi and cuda modules:

```
packages:
  openmpi:
    variants: +cuda fabrics=verbs
    buildable: false
    version: []
    target: []
    compiler: []
    providers: {}
    paths:
      openmpi@3.1.2%gcc@7.3.0: /opt/apps/gcc7_3/openmpi/3.1.2
    modules: {}

  cuda:
    modules:
      cuda@10.1: cuda/10.1
    buildable: false
    version: []
    target: []
    compiler: []
    providers: {}
    paths: {}

all:
    providers:
      mpi: [openmpi]
      blas: [netlib-lapack]
      lapack: [netlib-lapack]
```

The last section above sets defaults for all packages you'll installing with Spack – you might want to adjust those, or move them to an environment.

Fixing `config.guess` on longhorn

Spack has logic that will replace an outdated `config.guess` in a given package with a newer version. This comes in handy, because apparently the latest autoconf version is from 2012 and the `config.guess` that comes with it doesn't know about `ppc64le`. However, on longhorn, Spack will not find a newer `config.guess` in `/usr/share/automake*/` , hence things still don't work. To work around this, I hacked Spack like this:

```
diff --git a/lib/spack/spack/build_systems/autotools.py b/lib/spack/spack/build_
↪systems/autotools.py
index c21b8dad7..f8e8f64fe 100644
--- a/lib/spack/spack/build_systems/autotools.py
+++ b/lib/spack/spack/build_systems/autotools.py
@@ -133,11 +133,12 @@ def _do_patch_config_guess(self):
             if os.path.exists(path):
                 config_guess = path
         # Look for the system's config.guess
-        if config_guess is None and os.path.exists('/usr/share'):
-            automake_dir = [s for s in os.listdir('/usr/share') if
+        path_am = '/opt/apps/autotools/1.2/share'
+        if config_guess is None and os.path.exists(path_am):
+            automake_dir = [s for s in os.listdir(path_am) if
                             "automake" in s]
             if automake_dir:
-                automake_path = os.path.join('/usr/share', automake_dir[0])
+                automake_path = os.path.join(path_am, automake_dir[0])
```

(continues on next page)

(continued from previous page)

```
                path = os.path.join(automake_path, 'config.guess')
                if os.path.exists(path):
                    config_guess = path
```

2.4.5 Building WDMapp

You should be able to just follow the generic instructions from *Building WDMAPP*.

2.5 WDMApp on AiMOS at RPI

AiMOS is a 268 node IBM AC922 system with 2x IBM P9 and 6x NVIDIA V100 GPUs with 32GiB of memory each. More info is available on the AiMOS wiki:

https://secure.cci.rpi.edu/wiki/index.php?title=DCS_Supercomputer

Github access requires setting up the http(s) proxy

https://secure.cci.rpi.edu/wiki/index.php?title=Proxy

and you'll need to create an ssh key-pair for running MPI jobs

```
$ ssh-keygen # accept defaults
```

Warning: This will overwrite an existing `id_rsa` key pair.

2.5.1 Manual Install For Coupler Developers

XGC-Devel

Clone the repo

```
$ git clone https://github.com/wdmapp/XGC-Devel.git
$ cd XGC-Devel
$ git checkout rpi
```

Create a environment file with the following contents named `envAimosGcc74OpenMPI.sh`.

```
module use /gpfs/u/software/dcs-spack-install/v0133gcc/lmod/linux-rhel7-ppc64le/gcc/7.
↪4.0-1/
module load gcc/7.4.0/1 openmpi/3.1.4-mm5hjuq cmake/3.15.4-mnqjvz6
module load \
  adios/1.13.1-ev2p4am \
  adios2/2.5.0-mklg6ph \
  petsc/3.7.7-int32-hdf5+ftn-real-c-7ewou4w \
  fftw/3.3.8-b2oxdb5 \
  pkg-config/system-cyeqmxc
```

`source` the environment file

```
$ source envAimosGcc74OpenMPI.sh
```

Create a build directory `build-xgcDevel-aimosGcc74OpenMPI`, `cd` into it, and run `CMake`:

```
$ mkdir build-xgcDevel-aimosGcc74OpenMPI
$ cd !$
$ # specify the path to the XGC-Devel repo
$ cmake /path/to/xgc-devel/repo \
  -DCMAKE_CXX_COMPILER=g++ \
  -DCMAKE_C_COMPILER=gcc \
  -DCMAKE_Fortran_COMPILER=gfortran \
  -DXGC_USE_ADIOS1=ON \
  -DXGC_USE_ADIOS2=OFF \
  -DXGC_USE_CABANA=OFF \
  -DUSE_SYSTEM_PSPLINE=OFF \
  -DXGC_GENE_COUPLING=ON \
  -DBUILD_TESTING=OFF \
  -DCMAKE_INSTALL_PREFIX=$PWD/install
```

Run `make` to compile and link XGC:

```
$ make -j8
```

If all goes well the xgc binary will be created; `bin/xgc-es-cpp`.

GENE

Clone the repo

```
$ git clone https://github.com/wdmapp/gene.git
$ git checkout rpi
```

Create a environment file with the following contents named `envAimosGcc74OpenMPI.sh`.

```
module use /gpfs/u/software/dcs-spack-install/v0133gcc/lmod/linux-rhel7-ppc64le/gcc/7.
↪4.0-1/
module load gcc/7.4.0/1
module load openmpi/3.1.4-mm5hjuq
module load \
  cmake/3.15.4-mnqjvz6 \
  adios/1.13.1-zrrxpbi \
  adios2/2.5.0-rqsvxj4 \
  fftw/3.3.8-b2oxdb5 \
  netlib-scalapack/2.0.2-7bndnga \
  openblas/0.3.7-x7m3b6w \
  zlib/1.2.11-lpgvqh7 \
  hdf5/1.10.3-ftn-tgragps

export OMPI_CXX=g++
export OMPI_CC=gcc
export OMPI_FC=gfortran
```

`source` the environment file

```
$ source envAimosGcc74OpenMPI.sh
```

Create a build directory `build-gene-aimosGcc74OpenMPI`, `cd` into it, and run `CMake`:

```
$ mkdir build-gene-aimosGcc74OpenMPI
$ cd !$
$ # specify the path to the gene repo
$ cmake /path/to/gene/repo \
  -DCMAKE_Fortran_COMPILER=gfortran \
  -DCMAKE_CXX_COMPILER=g++ \
  -DCMAKE_C_COMPILER=gcc \
  -DGENE_USE_FUTILS=on \
  -DGENE_USE_ADIOS2=on \
  -DGENE_DIAG_PLANES=on \
  -DGENE_PERF=none \
  -DGENE_WDMAPP=on
```

Run `make` to compile and link GENE:

```
$ make -j8
```

If all goes well the gene binary will be created; `src/gene`.

Run

Clone the testcases repo (https://github.com/wdmapp/testcases) then follow the instructions in *run_1/README.md*:

https://github.com/wdmapp/testcases/blob/master/run_1/README.md

2.6 Setting up Spack

Spack is a generic package manager for HPC. We rely on it in the following to install WDMapp and its components. Setting up Spack is a one-time process on a given machine – if you already have a working Spack install, you should be able to use it. However, in practice there are plenty of ways that things can wrong, so we provide tested Spack setups for a selection of machines. Following our instructions makes sure that WDMapp is built with compatible compilers and machine-specific system packages (e.g., MPI, CUDA, etc.).

2.6.1 Installing Spack

Follow the instructions from the Spack Documentation.

```
$ git clone -b v0.15.4 https://github.com/spack/spack.git
```

Note: v0.15.4 is the latest spack stable version on 2020-10-20; newer versions will likely work but have not been tested. Using the default 'develop' branch is not recommended, as it does break sometimes and introduces a lot of package version churn if you try to track it.

Enable shell support for Spack.

```
# For bash/zsh users
$ export SPACK_ROOT=/path/to/spack
$ . $SPACK_ROOT/share/spack/setup-env.sh

# For tcsh or csh users (note you must set SPACK_ROOT)
```

(continues on next page)

(continued from previous page)

```
$ setenv SPACK_ROOT /path/to/spack
$ source $SPACK_ROOT/share/spack/setup-env.csh
```

2.6.2 Cloning the WDMapp package repo

Just clone the repository from github to the same machine that you just set up Spack on.

```
$ git clone git@github.com:wdmapp/wdmapp-config.git
```

2.6.3 Adding the WDMapp package repo to Spack

This will let Spack search the WDMapp repository for packages that aren't found in its builtin package repository.

```
$ spack repo add path/to/wdmapp-config/spack/wdmapp
==> Added repo with namespace 'wdmapp'.
```

Note: To update the wdmapp package repository to the latest, just run `git pull` in the directory where you cloned `wdmapp-config/`.

2.6.4 Machine-Specific Setup

Ubuntu 18.04

On Ubuntu 18.04, nothing special needs to be done, though installation can be sped up by adding a `packages.yaml` that teaches it about system-installed software so that it doesn't have to build everything from scratch.

2.7 Building WDMAPP

Note: In order to install the non-public packages that are part of the wdmapp metapackage, you will need SSH keys set up correctly with GitHub. This requires creating an ssh key on the machine you will be installing wdmapp (or use existing key if you have one) and adding the key to your GitHub account. If your SSH key has a passphrase (highly recommended), you will also need to run ssh-agent and add the key to ssh-agent with ssh-add. SSH Agent forwarding may also be an option, but be aware of the security trade offs and make sure it is supported on the target machine (in particular Summit and other OLCF machines do not allow agent forwarding). For details, see the GitHub documentation Connecting to GitHub with SSH.

2.7.1 Standard Installation

Building WDMapp can be done following the standard Spack way:

```
$ spack install wdmapp
```

Then, have a coffee and keep your fingers crossed.

The *wdmapp* packages is a metapackage ("BundlePackage" in Spack parlance) that pulls in proper versions and variants of GENE, XGC, the Coupler and TAU and all their dependencies to have everything in place to run a coupled simulation:

```
$ spack spec wdmapp
Input spec
--------------------------------
wdmapp

Concretized
--------------------------------
wdmapp@0.0.1%gcc@7.5.0+tau~xgc1_legacy arch=linux-ubuntu18.04-haswell
    ^coupler@master%gcc@7.5.0 build_type=RelWithDebInfo arch=linux-ubuntu18.04-haswell
    ^gene@coupling%gcc@7.5.0+adios2 build_type=RelWithDebInfo ~cuda cuda_arch=none␣
↪+diag_planes+futils perf=perfstubs +pfunit+wdmapp arch=linux-ubuntu18.04-haswell
    ^tau@develop%gcc@7.5.0+adios2~bgq+binutils~comm~craycnl~cuda+fortran~
↪gasnet+io+libdwarf+libelf~libunwind~likwid+mpi~ompt~opari~openmp+otf2+papi~pdt~
↪phase~ppc64le~profilepa
    ^xgc-devel@wdmapp%gcc@7.5.0~adios2 build_type=RelWithDebInfo ~cabana+coupling_
↪core_edge_gene arch=linux-ubuntu18.04-haswell
```

Then install effis:

```
$ spack install effis
```

2.7.2 Installing for development

Note: The following describes how to do development builds of gene (XGC can be done correspondingly) using *spack setup*. This way appears to be deprecated in favor of *spack dev-build*. It does work, but requires some workarounds and can subtly vary from how things are built using *spack install* or *spack dev-build*.

Most of the time, one needs to be able to specify the exact version of the code, make changes, etc. Spack can still help with development.

First of all, get a local clone of the git repository for, e.g., GENE. You probably already have one, so you can use that. Otherwise:

```
$ git clone git@gitlab.mpcdf.mpg.de:GENE/gene-dev.git
$ git checkout cuda_under_the_hood
```

The method described here relies on the cmake build of GENE, though you could choose to just use Spack to install dependencies for you, and then handle things manually as usual.

Next, create a build director and change into it:

```
$ cd gene-dev
$ mkdir build
$ cd build
```

Now, have Spack set up the build for you – but not actually do it:

```
$ spack setup gene@local
[...]
==> Generating spconfig.py
```

Note: Something is currently broken with Spack, which likely gives you `Error: 'SPACK_DEPENDENCIES'`. If that happens, you can work around it by setting `export SPACK_DEPENDENCIES=""` and trying agian.

The `spack setup` will install all required dependencies, and then creates `spconfig.py` file in the current directory. This script can be used as a replacement to the usual invocation of cmake.

```
$ ./spconfig.py .. # .. is the path to the sources
[...]
-- Generating done
-- Build files have been written to: /home/src/gene-dev/build
```

So then you're all set. Just call `make`.

```
$ make -j20
[...]
```

2.8 EFFIS

EFFIS is a workflow composition and execution framework, which enables efficient code coupling and provides a toolset for visualization and analysis. It is currently being developed within the ECP Whole Device Modeling project.

2.8.1 Workflow Composition in EFFIS

EFFIS jobs are composed with a YAML configuration file.

- `effis-compose.py <config file name>` builds the job
- `effis-submit <job directory>` submits the job to the queue.

Bulding and submitting the example Rhea job below looks something like:

```
$ effis-compose.py ~/testcases/run_1/rhea/run_1.yaml
$ effis-submit /gpfs/alpine/proj-shared/csc143/esuchyta/effis/rhea/wdmapp/xgc-gene-1
```

Detailed YAML Paramter Example

Below is a configuration file from our run repository for core-edge coupling on Rhea. To better itemize the discussion, it has been broken up into to several smaller units.

Some top-level keywords set where on the file system the job will run, as well as scheduler configurations.

```
jobname: xgc-gene  # Name of job given to scheduler (default = kittie-job, but␣
↪setting is recommended)
walltime: 3600     # Job walltime request in seconds (default = 3600)

# EFFIS required parameter: directory for job output, each code runs in separte␣
↪subdirectory
```

(continues on next page)

(continued from previous page)

```
rundir: /gpfs/alpine/proj-shared/csc143/esuchyta/effis/rhea/wdmapp/xgc-gene-1

# EFFIS required section: setup for the machine to run on
machine:
  name: rhea                  # rhea, summit, theta, cori, local
  queue: batch
  charge: csc143
  job_setup: run_1_setup.sh   # Shell script to run once job has started (on service
↪node)
```

Users can define their own variables, for referencing throughout the file. Deferencing looks like `${variable-name}`. (See further below.)

```
# User defined variables, see dereferencing in file-edits
period_1d: 1    # XGC output frequency for diagnosis.1D
period_3d: 1    # XGC output frequency for field3D
steps: 100      # Number of simulation steps to run
planes: 8       # Number of poloidal plans
rho: 4          # XGC radial parameter
```

Shell commands, copies, regular expression file search and replace (using Python `re` syntax) are available during job configuration. (See subsequent file sections as well.)

```
# Shell commands to run during job composition, i.e. before submission to scheduler
pre-submit-commands: ["mkdir coupling"]

# NOTE some EFFIS instructions can live in top-level or code level scope, e.g.
#   * pre-submit-commands
#   * copy, copy-contents
#   * link
#   * file-edit
```

`run` is the section where to specify the codes to run, along with their process decompositions.

```
# EFFIS required section: Codes to run and their setups
run:
```

Each code has a scope under `run`, where users can adjust settings specific to that code. ADIOS I/O groups can be configured, and matched where relevant for coupling.

Starting with `gene`.

```
# 'gene' (or others below) is just a label name, and will run in subdirectory name
↪gene/
gene:
  pre-submit-commands: ["mkdir out"]    # Like pre-submit-commands above
  processes: 16             # Number of MPI ranks
  processes-per-node: 16    # Number of MPI ranks per node
  cpus-per-process: 1       # Number of CPUs per MPI rank

  # File path to executable to run
  executable_path: /autofs/nccs-svm1_home1/esuchyta/spack/spack/opt/spack/linux-rhel7-
↪sandybridge/gcc-8.4.0/gene-app-coupling-pysy5qk373yqzlfjotayvxq3w4r4tjjh/bin/gene

  # Environment variables
  env:
    OMP_NUM_THREADS: 1
```

(continues on next page)

(continued from previous page)

```
    HDF5_USE_FILE_LOCKING: 'FALSE'

  # Files to copy
  copy:
    - ../GENE/parameters
    - ../GENE/XGC_map_circular_2020_new.h5
    - ../GENE/adios2cfg.xml
    - ../GENE/tracer_fast
    - ../GENE/profiles_ions
    - ../GENE/coupling.in

  # Files to edit (paramters is the main GENE configuration file)
  file-edit:
    parameters:
      - ['^\s*ntimesteps\s*=.*$', 'ntimesteps=${steps}']
      - ['^\s*n_planes\s*=.*$', 'n_planes=${planes}']

  # ADIOS groups are prefaced with leading .

  .density_coupling:
    output_path: density.bp
    adios_engine: SST

  # 'reads' matching for coupling reading
  .field_coupling:
    reads: xgc.field_coupling
```

`xgc` looks similar to `gene` but has different input files.

```
xgc:
  pre-submit-commands: ["mkdir restart_dir"]
  processes: 64             # Number of MPI ranks
  processes-per-node: 8     # Number of MPI ranks per node
  cpus-per-process: 1       # Number of CPUs per MPI rank

  # File path to executable to run
  executable_path: /autofs/nccs-svm1_home1/esuchyta/spack/spack/opt/spack/linux-rhel7-
↪sandybridge/gcc-8.4.0/xgc-devel-cmake-suchyta-z5m7gdz6miin7ypf6hpv2yuxszkn4rqd/bin/
↪xgc-es

  # Environment variables
  env:
    OMP_NUM_THREADS: 1
    HDF5_USE_FILE_LOCKING: 'FALSE'

  # Files to copy
  copy:
    - ../XGC/input
    - ../XGC/adioscfg.xml
    - ../XGC/adios2cfg.xml
    - ../XGC/petsc.rc
    - ../XGC/geqdsk_gene_comp_case5_fix.eqd
    - ../XGC/geqdsk_gene_comp_case5_fixed.eqd.node
    - ../XGC/geqdsk_gene_comp_case5_fixed.eqd.ele
    - ../XGC/den_gene_case5.prf
    - ../XGC/temp_gene_case5_fix.prf
    - ../XGC/perturbation.in
```

(continues on next page)

(continued from previous page)

```
    - ../XGC/ogyropsi_init_cond.bp

  # Files to edit (input is the main XGC configuration file)
  file-edit:
    input:
      - ['^\s*sml_mstep\s*=.*$', 'sml_mstep=${steps}']
      - ['^\s*sml_nphi_total\s*=.*$', 'sml_nphi_total=${planes}']
      - ['^\s*sml_grid_nrho\s*=.*$', 'sml_grid_nrho=${rho}']
      - ['^\s*diag_1d_period\s*=.*$', 'diag_1d_period=${period_1d}']
      - ['^\s*diag_3d_period\s*=.*$', 'diag_3d_period=${period_3d}']
      - ['^\s*adios_stage_3d\s*=.*$', 'adios_stage_3d=.true.']


  # ADIOS groups are prefaced with leading .

  .diagnosis.1d:
    output_path: xgc.oneddiag.bp
    adios_engine: BP4

  .field3D:
    output_path: xgc.3d.bp
    adios_engine: BP4

  .diagnosis.mesh:
    output_path: xgc.mesh.bp
    adios_engine: BP4

  # 'reads' matching for coupling reading
  .density_coupling:
    reads: gene.density_coupling

  .field_coupling:
    output_path: field.bp
    adios_engine: SST
```

One and two dimensional plotting can be turned on with special `run` keyword sections.

```
#  Plot all variable in XGC's diagnosis.1d that use psi as x-axis
plot-1D:
  x: psi
  data: xgc.diagnosis.1d

# Plot varables with same dimensions as XGC field3D's dpot on triangular mesh
plot-triangular:
  commandline_args:
    - rz               # Name in mesh file of variable for the nodes
    - nd_connect_list  # Name im mesh file for the node connectivity
    - dpot[0]          # Dimension setter

  .mesh:
    reads: xgc.diagnosis.mesh

  .plotter:
    plots: xgc.field3D
```

For clarity and completion, here is the full file.

```
jobname: xgc-gene  # Name of job given to scheduler (default = kittie-job, but␣
↪setting is recommended)
walltime: 3600     # Job walltime request in seconds (default = 3600)

# User defined variables, see dereferencing in file-edits
period_1d: 1   # XGC output frequency for diagnosis.1D
period_3d: 1   # XGC output frequency for field3D
steps: 100     # Number of simulation steps to run
planes: 8      # Number of poloidal plans
rho: 4         # XGC radial parameter

# EFFIS required parameter: directory for job output, each code runs in separte␣
↪subdirectory
rundir: /gpfs/alpine/proj-shared/csc143/esuchyta/effis/rhea/wdmapp/xgc-gene-1

# EFFIS required section: setup for the machine to run on
machine:
  name: rhea                  # rhea, summit, theta, cori, local
  queue: batch
  charge: csc143
  job_setup: run_1_setup.sh   # Shell script to run once job has started (on service␣
↪node)

# Shell commands to run during job composition, i.e. before submission to scheduler
pre-submit-commands: ["mkdir coupling"]

# NOTE some EFFIS instructions can live in top-level or code level scope, e.g.
#   * pre-submit-commands
#   * copy, copy-contents
#   * link
#   * file-edit


# EFFIS required section: Codes to run and their setups
run:

  # 'gene' (or others below) is just a label name, and will run in subdirectory name␣
↪gene/
  gene:
    pre-submit-commands: ["mkdir out"]   # Like pre-submit-commands above
    processes: 16            # Number of MPI ranks
    processes-per-node: 16   # Number of MPI ranks per node
    cpus-per-process: 1      # Number of CPUs per MPI rank

    # File path to executable to run
    executable_path: /autofs/nccs-svm1_home1/esuchyta/spack/spack/opt/spack/linux-
↪rhel7-sandybridge/gcc-8.4.0/gene-app-coupling-pysy5qk373yqzlfjotayvxq3w4r4tjjh/bin/
↪gene

    # Environment variables
    env:
      OMP_NUM_THREADS: 1
      HDF5_USE_FILE_LOCKING: 'FALSE'

    # Files to copy
    copy:
      - ../GENE/parameters
```

(continues on next page)

(continued from previous page)

```
      - ../GENE/XGC_map_circular_2020_new.h5
      - ../GENE/adios2cfg.xml
      - ../GENE/tracer_fast
      - ../GENE/profiles_ions
      - ../GENE/coupling.in

    # Files to edit (paramters is the main GENE configuration file)
    file-edit:
      parameters:
        - ['^\s*ntimesteps\s*=.*$', 'ntimesteps=${steps}']
        - ['^\s*n_planes\s*=.*$', 'n_planes=${planes}']

    # ADIOS groups are prefaced with leading .

    .density_coupling:
      output_path: density.bp
      adios_engine: SST

    # 'reads' matching for coupling reading
    .field_coupling:
      reads: xgc.field_coupling

  xgc:
    pre-submit-commands: ["mkdir restart_dir"]
    processes: 64            # Number of MPI ranks
    processes-per-node: 8    # Number of MPI ranks per node
    cpus-per-process: 1      # Number of CPUs per MPI rank

    # File path to executable to run
    executable_path: /autofs/nccs-svm1_home1/esuchyta/spack/spack/opt/spack/linux-
↪rhel7-sandybridge/gcc-8.4.0/xgc-devel-cmake-suchyta-
↪z5m7gdz6miin7ypf6hpv2yuxszkn4rqd/bin/xgc-es

    # Environment variables
    env:
      OMP_NUM_THREADS: 1
      HDF5_USE_FILE_LOCKING: 'FALSE'

    # Files to copy
    copy:
      - ../XGC/input
      - ../XGC/adioscfg.xml
      - ../XGC/adios2cfg.xml
      - ../XGC/petsc.rc
      - ../XGC/geqdsk_gene_comp_case5_fix.eqd
      - ../XGC/geqdsk_gene_comp_case5_fixed.eqd.node
      - ../XGC/geqdsk_gene_comp_case5_fixed.eqd.ele
      - ../XGC/den_gene_case5.prf
      - ../XGC/temp_gene_case5_fix.prf
      - ../XGC/perturbation.in
      - ../XGC/ogyropsi_init_cond.bp

    # Files to edit (input is the main XGC configuration file)
    file-edit:
      input:
        - ['^\s*sml_mstep\s*=.*$', 'sml_mstep=${steps}']
        - ['^\s*sml_nphi_total\s*=.*$', 'sml_nphi_total=${planes}']
```

(continues on next page)

(continued from previous page)

```
        - ['^\s*sml_grid_nrho\s*=.*$', 'sml_grid_nrho=${rho}']
        - ['^\s*diag_1d_period\s*=.*$', 'diag_1d_period=${period_1d}']
        - ['^\s*diag_3d_period\s*=.*$', 'diag_3d_period=${period_3d}']
        - ['^\s*adios_stage_3d\s*=.*$', 'adios_stage_3d=.true.']


    # ADIOS groups are prefaced with leading .

    .diagnosis.1d:
      output_path: xgc.oneddiag.bp
      adios_engine: BP4

    .field3D:
      output_path: xgc.3d.bp
      adios_engine: BP4

    .diagnosis.mesh:
      output_path: xgc.mesh.bp
      adios_engine: BP4

    # 'reads' matching for coupling reading
    .density_coupling:
      reads: gene.density_coupling

    .field_coupling:
      output_path: field.bp
      adios_engine: SST

  #  Plot all variable in XGC's diagnosis.1d that use psi as x-axis
  plot-1D:
    x: psi
    data: xgc.diagnosis.1d

  # Plot varables with same dimensions as XGC field3D's dpot on triangular mesh
  plot-triangular:
    commandline_args:
      - rz               # Name in mesh file of variable for the nodes
      - nd_connect_list  # Name im mesh file for the node connectivity
      - dpot[0]          # Dimension setter

    .mesh:
      reads: xgc.diagnosis.mesh

    .plotter:
      plots: xgc.field3D
```

2.8.2 Performance Monitoring with TAU

We have instrumented our codes, and to run XGC/GENE with TAU, `tau_exec` becomes the executable to run and the application is a command line argument to `tau_exec`. For example, a snippet for XGC might look like.

```
tau:  /autofs/nccs-svm1_home1/esuchyta/spack/wdmapp/rhea/spack/opt/spack/linux-rhel7-
↪sandybridge/gcc-8.4.0/tau-develop-ezg374unf3gephlxov5avmmagsplidn2/bin/tau_exec
xgc:  /autofs/nccs-svm1_home1/esuchyta/spack/wdmapp/rhea/spack/opt/spack/linux-rhel7-
↪sandybridge/gcc-8.4.0/xgc-devel-wdmapp-pvku3hgsn5pfzx3apmkn3fwrfefvu37a/bin/xgc-es
```

(continues on next page)

(continued from previous page)

```
run:
  xgc:
    executable_path: ${tau}
    commandline_args:
      - -T
      - mpi
      - -monitoring
      - -adios2
      - ${xgc}
      - -no_signal_handle
```

TAU expects the directory in which `tau_exec` lives to be `PATH`, so make sure to `spack load tau` in the `job_setup` file when using TAU, e.g. our example on Rhea.

EFFIS is able to generate plots of the performance data TAU collects, in a way similar to the one-dimensional plotting:

```
run:

  plot-tau-xgc:
    data: xgc.tau
    commandline_options:
      pattern: (BP4Writer|ADIOS_WRITE|MAIN_LOOP\s/)
      step: "MAIN_LOOP / Calls"
```

`pattern` uses Python re syntax to pick which variable to plot. `step` is which variable identifies the code's time step.

2.8.3 Enabling the Dashboard

To turn on the processing that packages up EFFIS images for use with the dashboard, one needs to configure a `dashboard` top-scope section in the configuration file.

```
base: wdmapp-1
rundir: /gpfs/alpine/proj-shared/csc143/esuchyta/effis/rhea/wdmapp/${base}
adios-nompi: /autofs/nccs-svm1_home1/esuchyta/spack/wdmapp/rhea/spack/opt/spack/linux-
↪rhel7-sandybridge/gcc-8.4.0/adios2-2.6.0-k62srrf7btzanzeattmk35orgqd6uyvn
dashboard:
  use: true
  shot_name: ${base}
  run_name:  run-1
  http: /ccs/wwwproj/phy122/esuchyta/wdmapp-dashboard/shots
  env:
    ADIOS: ${adios-nompi}/lib/python3.7/site-packages
```

`http` is a directory that is web accessible for remote download. At OLCF, there is one such area, and it is only accessible from the service/login nodes, where MPI does not work. This is why an ADIOS2 build without MPI is needed, which can be built with:

```
$ spack install adios2 -mpi +python
```

`shot-name` and `run-name` are both text labels, where `run-name` is meant to allow multiple restarted runs in the same shot.

Deploying the Remote Dashboard

Instructions for how to deploy a run an instance of the dashboard that connects to this data can be found on in the eSimMon documentation. In short, a monitoring service will given the web address for the `http` directory, pull new data when it becomes available, and then display the images thorugh a web server that multiple uers can connect to.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search