The TECA, Toolkit for Extreme Climate Analysis, User's Guide

Lawrence Berkeley National Lab

November 16, 2015

Contents

1 Installing and Running TECA

1.1 Download

1.1.1 Binaries

1.1.2 Sources

1.2 Build and Installation

1.2.1 TECA

Supported Compilers Building TECA requires a C++11 compiler. On Unix like systems this is GCC 4.9(or newer), or LLVM 3.5(or newer). Windows MS Visual Studio C++ currently does not fully support C++11. The following dependencies are optional, however functionality will be reduced if they are not present.

- CMake for configuring the build
- NetCDF for the CF reader
- MPI for distributed parallel operation
- Boost program_options for the command line applications
- VTK for the ability to save mesh based data for later visualization in ParaView or VisIt

The location of various dependencies should be passed in during configuration if they are in non-standard locations. It's is critical that compiler and C++ library versions match across dependencies, especially Boost.

Compiling TECA on a Workstation This is an example of compiling on a work station, with Boost, MPI, NetCDF and VTK features enabled:

```
#!/bin/bash

cmake \
    -DCMAKE_CXX_COMPILER=`which clang++` \
    -DCMAKE_C_COMPILER=`which clang` \
    -DCMAKE_BUILD_TYPE=Release \
    -DNETCDF_DIR=/work/apps/netcdf/4.3.3.1/ \
    -DVTK_DIR=/work/apps/vtk/next/lib/vtk-6.3/cmake \
    -DBOOST_ROOT=/work/apps/boost/1.58.0 \
    $*

make -j8 && make -j8 install
```

Compiling TECA on a Cray The following shows how TECA is compiled on NERSCs Cray XC30 Edison with Boost, MPI, and NetCDF.

```
#!/bin/bash

module load cmake/3.0.0
module swap PrgEnv-intel PrgEnv-gnu
module load netcdf/4.3.3.1
module load boost/1.58.0

export XTPE_LINK_TYPE=dynamic
LIB_EXT=so

NETCDF=/usr/common/graphics/netcdf/4.3.3.1
BOOST=/usr/common/graphics/boost/1.58.0/

SMA=/opt/cray/mpt/7.2.1/gni/sma/lib64
MPT=/opt/cray/mpt/7.2.1/gni/mpich2-gnu/49/lib
RCA=/opt/cray/rca/1.0.0-2.0502.57212.2.56.ari/lib64
ALPS=/opt/cray/alps/5.2.3-2.0502.9295.14.14.ari/lib64
XPMEM=/opt/cray/xpmem/0.1-2.0502.57015.1.15.ari/lib64
DMAPP=/opt/cray/dmapp/7.0.1-1.0502.10246.8.47.ari/lib64
PMI=/opt/cray/pmi/5.0.6-1.0000.10439.140.2.ari/lib64
UGNI=/opt/cray/ugni/6.0-1.0502.10245.9.9.ari/lib64
UDREG=/opt/cray/udreg/2.3.2-1.0502.9889.2.20.ari/lib64
WLM=/opt/cray/wlm_detect/1.0-1.0502.57063.1.1.ari/lib64
ATP=/opt/cray/atp/1.8.2/libApp

CXXCOMP=`which g++`
CCOMP=`which gcc`

cmake \
  -DCMAKE_CXX_COMPILER=$CXXCOMP \
  -DCMAKE_C_COMPILER=$CCOMP \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_EXE_LINKER_FLAGS_RELWITHDEBINFO="$ATP/libAtpSigHandler.$LIB_EXT $ATP/libAtpSigHCommData.a -Wl,--
  -DCMAKE_EXE_LINKER_FLAGS_DEBUG="$ATP/libAtpSigHandler.$LIB_EXT $ATP/libAtpSigHCommData.a -Wl,--undefined
  -DMPI_CXX_COMPILER=$CXXCOMP \
  -DMPI_C_COMPILER=$CCOMP \
  -DMPI_CXX_LIBRARIES="" \
  -DMPI_C_LIBRARIES="-Wl,--start-group;$MPT/libmpichcxx.$LIB_EXT;$SMA/libsma.$LIB_EXT;$PMI/libpmi.$LIB_EXT
  -DMPI_INCLUDE_PATH=$MPT/../include \
  -DMPIEXEC=$APRUN/bin/aprun \
  -DNETCDF_DIR=$NETCDF \
  -DBOOST_ROOT=$BOOST \
  -DCMAKE_INSTALL_PREFIX=/usr/common/graphics/teca/1.0 \
  ../teca

make -j 8 && make -j 8 install
```

1.2.2 Prerequisites

Dependencies can be installed from a package manager where convenient. However, note that particularly with Boost, compiler and stdlib used to build Boost must match that used with TECA. When using cmake on systems with multiple compilers one must consistently specify CMAKE_CXX_COMPILER and CMAKE_C_COMPILER and/or export CC and CXX environment variables.

CMake TECA builds are configured with CMake. Version 2.8.12 or newere is required. Installing via your system's package manager is recommended.

MPI MPI is required for distributed parallel operation. It's recommended to use the package management system on your OS to install MPI.

NetCDF A standard make, make install suffices. It is fine to disable NetCDF 4 features. Replace gcc and g++ with your compiler, for example clang and clang++ on Apple.

```
export CC=‘which gcc‘
export CXX=‘which g++‘
wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4.3.3.1.tar.gz
tar xzfv netcdf-4.3.3.1.tar.gz
cd netcdf-4.3.3.1
./configure --disable-netcdf-4 --prefix=/work/apps/netcdf/4.3.3.1
make -j2 && make -j4 install
```

Boost When possible use the package manager to install Boost. However note that the compiler and stdlib version used to build Boost **needs to match exactly** the compiler and stdlib used to build TECA. This may necessitate a stand alone Boost install.

On Linux if the compiler yo are using is in the PATH then a simple install suffices.

```
#!/bin/bash
curl -L <HTTP LINK TO BOOST PACKAGE> -o boost.tar.gz
tar xzfv boost.tar.gz
cd boost_1_58_0/
./bootstrap.sh --prefix=/Users/bloring/apps/
./b2 -j4 cxxflags=’’-std=c++11’’ install
```

However, on Apple, when using clang you must specify toolset and flags when building Boost:

```
#!/bin/bash
./b2 -j4 toolset=clang cxxflags="-stdlib=libc++" linkflags="-stdlib=libc++" install
```

VTK

```
#!/bin/bash
git clone git://VTK.org/VTK.git
mkdir vtk_build
cd vtk_build
cmake \
  -DCMAKE_CXX_COMPILER=‘which g++‘ \
  -DCMAKE_C_COMPILER=‘which gcc‘ \
  -DCMAKE_INSTALL_PREFIX=/work/apps/VTK/next \
  ../VTK
make -j4 && make -j4 install
```

1.3 Running the Pre-packaged Applications

See –help and –full_help command line arguments for the application in question.

1.3.1 AR Detector

Running on a Cray make a qsub file and submit a job. Here is an example:

```
#!/bin/bash -l
#PBS -q premium
#PBS -l mppwidth=2400
#PBS -l walltime=01:00:00
#PBS -N teca_tmq
#PBS -j oe

TECA_HOME=/usr/common/graphics/teca/1.0

cd $PBS_O_WORKDIR
aprun -n 200 -N 2 -S 1 \
    ${TECA_HOME}/bin/teca_ar_detect \
        --water_vapor_file_regex TMQ_cam5_1_amip_run2'.*nc' \
        --water_vapor_var TMQ \
        --n_threads 12 \
        --results_file tmq_ar.csv
```

1.3.2 TC Detector

1.3.3 ETC Detector

1.4 Algorithm Library

1.4.1 Input and Data Readers

1.4.2 Detectors

1.4.3 Analsyis

1.4.4 Re-meshing

1.4.5 Output and Data Writers

1.5 Python Scripting

[back to contents]

2 TECA Framework Design and Architecture

2.1 The Pipeline Pattern and the Algorithm Abstraction

2.2 Information and Data Flow in the Pipeline

2.3 Algorithm Abstraction

2.4 Metadata-structures

2.5 Data-structures

2.5.1 Mesh Based Data

2.5.2 Tabular Data

[back to contents]

2.6 Examples

[back to contents]

3 Contributing Code to TECA

[back to contents]

3.1 Github Workflow

3.2 Coding Standard

3.3 Regression Testing

3.4 Writing an Algorithm

3.5 Algorithm Template

3.6 Adding a Dataset

3.7 Porting an Existing Algorithm

4 Publications

[back to contents]