DGM Installation Guide

Date: Mon Feb 17 02:37:44 MST 2014 (2014-02-17)
Revision: 1.2

Contents

1 Building this Documentation

If you have `Docutils` available on your system, you can build the HTML version of this documentation using the command:

```
% rst2latex.py --documentoptions=letter \
  --stylesheet=fullpage,parskip Install.rst Install.tex
% pdflatex Install
```

Likewise, you can build an HTML version of this documentation using:

```
% rst2html.py Install.rst Install.html
```

You can find `Docutils` at `http://docutils.sourceforge.net`.

2 DGM Quick Build and Install

These are the basic steps to build DGM assuming,

- the Third Party Libraries (TPL) are already setup in `~/local` (if not see details below in the Third Party Libraries section),
- you are performing a default installation, and
- you are using the GCC compilers.

Additional and customization information on each of the steps is given in the detail sections below.

2.1 Obtaining DGM

There are three basic ways to accessing a copy of DGM: download a copy from the svn repository:

```
% cd /path/to/distribution/location
% svn checkout svn+ssh://software.sandia.gov/svn/private/dgm/trunk dgm
```

obtain a tarball from a DGM developer, or use a copy installed in a "projects" directory (e.g., `/home/dgm`) by another developer or by a nightly build system, e.g., Jenkins.

2.2 Setting Up Environment

Get a copy of Jenkins environment setup script, and modify it for your directory structure and machine. Do not directly edit this file, as it is used by Jenkins for builds; make a personal copy. For more details on setting up your environment, see the detail information below.:

```
% export DGM_DIST=/path/to/distribution/location/dgm
% cp $DGM_DIST/regression/jenkins/dgm-env-setup-<machine>.bsh \
     $HOME/dgm-env-setup-<machine>.bsh
# vi and set DGM_DIST, DGM_TPL, TRILINOS_DIST, NETCDF_DIR, ...
% source $HOME/dgm-env-setup-<machine>.bsh
```

2.3 Building and Installing DGM

```
% cd $DGM_DIST
% ./make_dgm -cisp                # build serial and parallel versions
% ./make_dgm -cispd               # and install in ~/local/dgm
```

Or to build with Trilinos:

```
% ./make_dgm -cispdT
% ./make_dgm -m                   # build the Doxygen documentation
```

You should now have executables in the default location `$DGM_TPL/dgm`.

2.4 Testing DGM

```
% cd $DGM_DIST/runs
% runtests.py
```

For instructions for building DGM on Cray systems, please see the section near the end of this document entitled Cray Instructions.

3 Detailed Information

3.1 Obtaining Software

For external users/developers or individuals without access to the DGM repository, please contact

Scott Collis
Computational Science and Mathematics
Sandia National Laboratories
sscoll@sandia.gov

to obtain a DGM download package, and follow the unpacking instructions.

For internal users/developers with access to the DGM repository, a copy of DGM can be obtained via an svn checkout:

```
% cd /path/to/distribution/location
% svn checkout svn+ssh://software.sandia.gov/svn/private/dgm/trunk dgm
% export DGM_DIST=/path/to/distribution/location/dgm
```

3.2 Setting Up Environment

Fill-in details.

3.3 Building DGM

Assuming that you have the required Third-Party Libraries (TPL) installed and available to DGM (see the section below on Third-Party Libraries), you can build DGM simply by executing autobuild:

```
% cd $DGM_DIST
% ./make_dgm -cisp            # build serial and parallel versions
% ./make_dgm -cispd           # and install in ~/local/dgm
```

Or to build with Trilinos:

```
% ./make_dgm -cispT
% ./make_dgm -cispdT

% ./make_dgm -m               # build the Doxygen documentation
```

or:

```
% cd $DGM_DIST
% ./autobuild                 # script to do above plus an svn update
                              # and build DGM/Trilinos
```

This will build serial, parallel, optimized and debug versions of DGM and its utilities (via `make_dgm`). autobuild will also build versions of DGM that use Trilinos (via `make_dgm -T`). Lastly it will generate doxygen webpages, and pdf User's Guide.

If you wish to build individual versions, use `make_dgm -h` to see the usage. By default, `make_dgm` will build an optimized serial version. `make_dgm -sp` will build an optimized serial and parallel versions.

If you want to build with shared libraries and Trilinos, then you would:

```
% cd $DGM_DIST
% ./make_dgm -ispyT
```

Note that on Linux, this requires that you have built (or have available to you) shared library versions of all TPLs. On MacOS-X (Darwin), everything is always dynamic anyway so this shouldn't be an issue.

3.4 Installing DGM

To install DGM in the default location (`$HOME/local/dgm` or `$DGM_TPL/dgm`), you can use the `make_dgm` script with the `-i` flag. This also rebuilds DGM before installing:

```
% make_dgm -cisp
```

and/or:

```
% make_dgm -cispd
```

This will install both the serial and parallel builds of DGM in the default locations:

```
$DGM_TPL/dgm/{serial,mpi}/{opt,debug}
```

If `$DGM_HOME` is defined, then the builds will go to `$DGM_HOME/{serial,mpi}/{opt,debug}`

The user may wish to add to their environment (`.cshrc` or `.bashrc`) the path to the DGM executables (serial and parallel) and utilities, e.g.,:

```
% export DGM_HOME=$DGM_TPL/dgm
% export PATH=$DGM_HOME/serial/opt/bin:$DGM_HOME/mpi/opt/bin:$PATH
```

The executables, headers and the DGM library are copied to the corresponding `bin/`, `include/` and `lib/` directories, e.g.,:

```
$DGM_HOME/serial/opt/bin
$DGM_HOME/serial/opt/include
$DGM_HOME/serial/opt/lib
$DGM_HOME/mpi/opt/bin
$DGM_HOME/mpi/opt/include
$DGM_HOME/mpi/opt/lib
```

and for debug:

```
$DGM_HOME/serial/debug/bin
$DGM_HOME/serial/debug/include
$DGM_HOME/serial/debug/lib
$DGM_HOME/mpi/debug/bin
$DGM_HOME/mpi/debug/include
$DGM_HOME/mpi/debug/lib
```

The `DGM_HOME` environment variable can be modified by the user to point to whichever set of executables and utilities.

3.5 Customizing the Installation

If you want full control over the installation location, you can override all of this by setting the environment variable, `DGM_DEST`. To install the executables to a custom location, you would do something similar to the following to reproduce the default installation:

```
% env DGM_DEST=$DGM_HOME/serial/opt make_dgm -is
% env DGM_DEST=$DGM_HOME/serial/debug make_dgm -isd
% env DGM_DEST=$DGM_HOME/mpi/opt make_dgm -ip
% env DGM_DEST=$DGM_HOME/mpi/debug make_dgm -ipd
```

Note that using `DGM_DEST` requires that you manage the differences between serial, mpi, debug and optimized build locations yourself. An alternative is to set `DGM_HOME` such as:

```
% env DGM_HOME=$HOME/my_local make_dgm -isp
```

and this would install using the "normal" installation sub-paths within this `DGM_HOME` location.

You may also using the `make install` build option and specify the destination directly to the build system using:

```
% cd $DGM_DIST/GCC
% make install DGM_DEST=/your/custom/destination/serial/opt
% cd $DGM_DIST/GCCp
% make pinstall DGM_DEST=/your/custom/destination/mpi/opt
```

For a Trilinos-enabled build, the typical user can type:

```
% make_dgm -isp -t $DGM_TPL/trilinos
```

to build and install Trilinos-enabled version of DGM as long as you follow the structure of your Trilinos installs that I do which is::

```
$DGM_TPL/trilinos/mpi/opt
$DGM_TPL/trilinos/mpi/debug
$DGM_TPL/trilinos/serial/opt
$DGM_TPL/trilinos/serial/debug
```

3.6 Testing DGM Build

To test the build, you will need to run the following:

```
% export DGM_HOME=$DGM_TPL/dgm
% export PATH=$DGM_HOME/serial/opt/bin:$DGM_HOME/mpi/opt/bin:$PATH
% cd $DGM_DIST/runs
% runtests.py
```

If successful you should see output similar to:

```
Fri Dec 10 09:18:45 2010
Test Results from Directory: $DGM_DIST/runs
Total number of test(s): 29
------------------------------------------------------------------------
  pass         1.96s  1d.tst
...
  skipped      0.05s  poisson/poisson.tst
------------------------------------------------------------------------
 Pass: 25    Fail: 0    Skipped: 4    Total: 29

Total Runtime:     256.13s
```

Skipped tests are expected as some tests are in development or take too long to run for a simple installation test. So long as there are no failing tests, the build is good.

If you are not using a Trilinos enabled build of DGM, then you should instead do:

```
% runtests.py -K Trilinos
```

and this will make sure to exclude Trilinos-enabled tests.

3.7 Building DGM with ROL

First get a copy of ROL, e.g. from the Trilinos repository, (See Trilinos install section, but for reference):

```
% git clone https://github.com/trilinos/Trilinos.git
% cd Trilinos/packages/rol
% python rol_install.py $DGM_TPL/trilinos/mpi/opt/include
% python rol_install.py $DGM_TPL/trilinos/serial/opt/include
```

Also set DGM_USE_ROL:

```
% export DGM_USE_ROL=1
```

or:

```
% make_dgm -D DGM_USE_ROL
```

First get a copy of ROL, e.g. from the rol-only repository,:

```
% cd $DGM_TPL
% git clone software.sandia.gov:/git/rol rol-only
```

Then install in both the parallel and serial include directories.:

```
% cd rol-only/rol
% export ROL_HOME=$DGM_TPL/trilinos/mpi/opt
% ./install.sh
% export ROL_HOME=$DGM_TPL/trilinos/serial/opt
% ./install.sh
```

3.8 Building DGM with peopt

First one needs to obtain a copy of peopt, e.g.,:

```
% cd /path/to/distribution/location
% svn checkout --username guest https://teamforge.sandia.gov/svn/repos/peopt/trunk peo
% cd peopt
% setenv PEOPT_HOME /path/to/place/of/peopt/install
% mkdir build
% cd build
% cmake -DCMAKE_INSTALL_PREFIX=$PEOPT_HOME ../src
% make install
```

Then one can proceed with the normal build and installation steps, e.g.:

```
% make_dgm -isp
% make_dgm -ispd
```

So, by making sure that PEOPT_HOME is defined, you will automatically get pdgm_tropt.exe. One may wish to add the PEOPT_HOME environment variable to their .bashrc for subsequent builds, or alternatively use:

```
% PEOPT_HOME=/path/to/place/of/peopt/install make_dgm -isp
% PEOPT_HOME=/path/to/place/of/peopt/install make_dgm -ispd
```

Note that peopt does not currently support a serial build but the build system is smart enough just to ignore a request for a serial build.

3.9 Building with JSON

In order to enable JsonCpp, which is required for PEOPT, we must turn on the JsonCPP compile flag:

```
% export USE_JSON=1
```

Then one can proceed with the normal build and installation steps for DGM, e.g.:

```
% cd $DGM_DIST
% make_dgm -isp
% make_dgm -ispd    # optional debug builds
```

So, by making sure that PEOPT_HOME is defined, you will automatically get pdgm_peopt.exe. One may wish to add the PEOPT_HOME environment variable to their .bashrc for subsequent builds, or alternatively use:

```
% cd $DGM_DIST
% env USE_JSON=1 PEOPT_HOME=/path/to/distribution/location/peopt make_dgm -sp
% env USE_JSON=1 PEOPT_HOME=/path/to/distribution/location/peopt make_dgm -spd
```

Note that peopt does not currently support a serial build but the build system is smart enough just to ignore a request for a serial build.

4 Required Third-Party Libraries

IF you are on machine that has a DGM project directory (e.g., `/home/dgm`), you should be able to link against the Third-Party Libraries (TPL) in the project directory (e.g., `/home/dgm/local`). There may be a link_local script there to help you.

Anotherwise you may need to build the TPLs yourself. The following should be helpful in this case.

For this build to be successful, DGM requires access to several Third-Party Libraries (TPL). These are:

ATLAS: Automatically tuned versions of BLAS and (some) LAPack

Boost: smart pointers and optionally random number generators

Trilinos: For linear, nonlinear, and optimization solvers and, optionally smart pointers. Trilinos is optional, but is highly recommended and in future versions of DGM it may be required.

By default, DGM is configured to use Boost, ITL, GSL and FFTW and unless specified directly, these are assumed to be installed in a directory called `$HOME/local` where `$HOME` is the users home directory.

If you don't wish to have a `$HOME/local` directory, the build system supports two optional approaches. The first is to specify the TPL installation directory using an environmental variable, e.g.:

```
% cd $DGM_HOME/SERIAL
% make dgm DGM_TPL=<put your TPL location here>
```

If you have set `DGM_TPL` in your environment, the `make_dgm` script will automatically try to use this version.

Another approach is to create a symbolic link in your build directory to the TPL installation directory. For example:

```
% cd $DGM_HOME/SERIAL
% ln -s <your TPL installation directory> local
% make dgm
```

Note that (excepting JsonCpp) no third-party libraries are distributed with DGM so that you are responsible for downloading and installing any TPL that you wish to use in accordance with is associated license.

4.1 ATLAS

In reality, DGM doesn't require ATLAS, but instead needs working versions of BLAS and several routines from LAPACK. However, I perfer to get my BLAS and LAPACK from ATLAS as I have usually found it to provide greater performance for DGM operations relative to a default BLAS and most vendor-supplied BLAS/LAPACK. This is because, ATLAS, when built, is automatically tuned for the given architecture and for a variety of data sizes. For the sometimes small data sizes used by DGM, this proves advantageous over vendor BLAS which are typically optimized for large vector/matrix sizes.

- Get version of ATLAS from http://sourceforge.net/projects/math-atlas/files/Developer (unstable)

```
% cd $DGM_TPL   # (e.g., $HOME/local)
% rm atlas
% tar --bzip2 -xf atlas3.9.21.tar.bz2
%  (or 'bunzip2 -c atlas3.9.21.tar.bz2 | tar xfm -')
% mv ATLAS atlas-3.9.21
```

- Checkout errata at http://math-atlas.sourceforge.net/errata.html. Topics I found interesting
  - How about C++ header files for the C interfaces?
  - ATLAS install fails on G4 when using gnu gcc instead of Apple's gcc
  - Search doesn't work with cpu throttling.
  - Building a complete LAPACK library
  - Problems with linking/missing LAPACK routines on OS X
- Turn off the CPU throttling (not necessarily recommended)
  - on Linux (fedora 14) CPU throttling must be turned off otherwise 'configure' will fail. The ATLAS notes suggest the following command that allowed the configure to proceed.:

    ```
    % cpufreq-selector -g performance
    ```

  - For a Mac OSX, I found the following https://discussions.apple.com/message/3480072
  - Copy the folder named `IOPlatformPluginFamily.kext` which is in `/System/Library/Extensions` and put it on a thumbdrive or other external media.
  - Move the entire folder named above to the trash and empty it.
  - Then restart.
  - You can drag the folder back to its location and restart when you want to reactivate speed step. You may need to repair permissions using Disk Utility.
- Create build directory:

```
% cd atlas-3.9.21
% mkdir Build
```

- Configure ATLAS for build, assuming
  - that we do not want/need a full LAPACK library and Fortran APIs.
  - that CPU throtting has been turned off.

```
% cd Build
%
% # Machine is not loaded (no other processes running) (-D c -DWALL)
% # 64-bit architecture (-b 64)
% ../configure -v 2 -b 64 --dylibs -D c -DWALL \
%  --prefix=$DGM_TPL/atlas-3.9.21-install
%
% # Want to build dynamic/shared libraries ( --dylibs -Fa alg -fPIC)
% ../configure -b 64 --dylibs -Fa alg -fPIC \
%  --prefix=$DGM_TPL/atlas-3.9.21-install
%
% # For MacBook Pro Snow Leopard (64 bit)
```

```
% ../configure --nof77 -b 64 --dylibs \
%  --prefix=$DGM_TPL/atlas-3.9.21-install
%
% # For Mac Leopard (32 bit) (ictinus)
% ../configure --nof77 -b 32 --dylibs
%  --prefix=/Users/dgm/local/atlas-3.9.21-install
%
% make build        # This took about 2-3 hours on Mac OS 10.6
% cd lib
% make cdylib       # Not needed if --dylibs or --share used with configure
% cd ..
% make check        # Will fail if not built with F77
% make time         # Will fail if not built with F77
% make install      # For Macs it will try to install *.so and it will not
%                     find them.  This is OK.
```

- Set links to ATLAS:

```
% cd $DGM_TPL
% ln -s atlas-3.9.21-install atlas
```

- On fedora 18 with the standard ATLAS libs installed, the makefile attempts to link with `libatlas_clapack` which does not exist. This seems like an error; a work around was to create a symbolic link:

```
% ln -s /usr/lib64/atlas/libclapack.so /usr/lib64/atlas/libatlas_clapack.so
```

- For RHEL6, see previous note, with the exception that the target library may be `/usr/lib64/atlas/libclapack.so.3` or similar. Also, the make script does not seem to be picking up `LD_LIBRARY_PATH`, so the link may have to be created in `$HOME/local/atlas/lib`.
- If using ATLAS 3.11.11, you may get the use error: `ATL_gemm.c:(.text+0x13c): undefined reference to ATL_sammm` This is a known issue in this version of ATLAS. No fix yet. You may want to try an older, more stable version.

4.2 OpenMPI

To run in parallel, DGM requires some form of MPI. The following instructions are for OpenMPI, but you may also use MPICH2 as well as vendor provided versions of MPI.

1) Download the latest version from http://www.open-mpi.org/software/ompi/v1.6/
2) In the directory where you want to put the OpenMPI installation, execute:

```
% cd $DGM_TPL   (e.g., $HOME/local)
% tar zxf openmpi-1.6.3.tar.gz
```

3) Configure OpenMPI:

```
% cd openmpi-1.6.3
% ./configure --prefix=$DGM_TPL/openmpi-1.6.3_build
```

4) Build OpenMPI:

```
% make all install
```

5) Set link to new OpenMPI:

```
% cd $DGM_TPL
% ln -s openmpi-1.6.3_build openmpi
```

6) You will likely need to do a clean build (at least if not rebuild everything):

```
% cd $DGM_DIST
% make_dgm -cisp
```

7) If you are on a laptop or workstation that is using VPN you may find that VPN blocks the TCP communication used by the Out-of-Band communication during startup i.e. `MPI_Init`. In this case, you may want to add the following to your `.bash_profile`. First set the main transport to be shared memory and don't use TCP:

```
% export OMPI_MCA_blt="^tcp,sm,self"
```

Now make sure that Out of Band communication (i.e. setup) uses localhost:

```
% export OMPI_MCA_oob_tcp_if_include="127.0.0.0/24"
```

This last variable can also be set from the `mpiexec` command line using:

```
% mpiexec -mca oob_tcp_if_include 127.0.0.0/24 pdgm.exe root
```

The important thing here is that OpenMPI tries to use the TCP stack from your current host for setup and VPN likely blocks that host resulting in a timeout. By setting OOB TCP to use the localhost, you should be able to run.

4.3 Boost

While may Boost capabilities are fully templated and available soley in C++ headers, DGM makes use of several Boost packages that must be built into libraries before they can be used. This includes `boost::mpi`, `boost::filesystem`, and `boost::random`. The following instructions describe how to build the required boost libraries on a typical Linux-like system.

1) Download the latest version from http://www.boost.org/

2) In the directory where you want to put the Boost installation, execute:

```
% cd $DGM_TPL   (e.g., $HOME/local)
% tar --bzip2 -xf boost_1_55_0.tar.bz2
```

3) Configure boost with libraries:

```
% cd boost_1_55_0
% export BOOST_DIST=/Users/ccober/local/boost_1_55_0
% ./bootstrap.sh \
%  --with-libraries=system,filesystem,random,signals,serialization,mpi,program_opti
%  --prefix=$DGM_TPL/boost_1_55_0
```

Building the boost::mpi library requires you to modify file `$BOOST_DIST/tools/build/v2/user-config.jam` to include the line `using mpi ;` Note: if you want to specify the compiler, such as in `$DGM_TPL/openmpi`, then the line you add should be `using mpi : <full-path-to-mpicxx> ;` where you fill in the full path name, e.g., `using mpi : /Users/ccober/local/openmpi/bin/mpicxx ;`

Note, as of Boost v1.56.0 the location of `user-config.jam` has changed and is now at:

```
$BOOST_DIST/tools/build/example/user-config.jam
```

You now need to copy that example to your distribution directory:

```
% cp $BOOST_DIST/tools/build/example/user-config.jam $BOOST_DIST
```

And then edit the file as described above. Note that you also have to set `BOOST_BUILD_PATH=$BOOST_DIST` so that your `user-config.jam` can be located

4) Build boost:

```
% ./b2 install
```

or with a custom user-config.jam you would do:

```
% env BOOST_BUILD_PATH=$BOOST_DIST ./b2 install
```

so that the build system will find your `user-config.jam`.

5) Set link to new boost:

```
% cd $DGM_TPL
% ln -s boost_1_55_0 boost
```

6) You will likely need to do a clean build:

```
% cd $DGM_DIST
% make_dgm -cisp
```

7) Need to add the boost library path to your environment:

```
% export LD_LIBRARY_PATH=$DGM_TPL/boost/lib:$LD_LIBRARY_PATH
```

or for Darwin (Mac OS-X) you would set:

```
% export DYLD_LIBRARY_PATH=$DGM_TPL/boost/lib:$DYLD_LIBRARY_PATH
```

8) Under Mac OS X 10.10 (El Capitan) and later, security settings have been changed so that you can no longer set `DYLD_LIBRARY_PATH` and expect that it will be propogated to a sub-shell. This is inconvient for the DGM testing system in particular as we often use subshells to execute runs.

So, we need use `otool -L dgm.exe` to see where the OS is trying to find dynamic libraries and if the paths are not correct (or incomplete) you need to use `install_name_tool -id path` on the dylib to set it correct. For boost, I've found that it doesn't set the install name in the dylibs correctly so that you need to do that by hand. If I can figure out how to make the boost build system do it, I'll add that in here.

5 Optional Third-Party Libraries

The following third-party libraries are optional but can be included to actively significant enhanced capabilities within DGM such as implicit time integration, iterative linear and nonlinear solvers, and the ability to translate ExodusII files to DGM formats.

5.1 HDF5

HDF5 is a hierarchical file format that is designed for scientific data. Both Netcdf and ExodusII formats require HDF5 to build.

1) Download hdf5-1.8.9 from http://www.hdfgroup.org/ftp/HDF5/current/src/hdf5-1.8.9.tar.gz

2) Move and uncompress HDF5 in build location (e.g., `$HOME/local` or `$DGM_TPL`):

```
% mv hdf5-1.8.9.tar.gz $DGM_TPL/.
% cd $DGM_TPL   # (e.g., $HOME/local)
% tar zxvf hdf5-1.8.9.tar.gz
```

3) Build HDF5 twice, once serial, once parallel:

```
% cd hdf5-1.8.9
% ./configure --prefix=$DGM_TPL/hdf5-1.8.9/serial/opt
```

or

```
% CC=/Users/ccober/local/openmpi/bin/gcc-mp-4.4 \
  CXX=/Users/ccober/local/openmpi/bin/g++-mp-4.4 \
  FC=/Users/ccober/local/openmpi/bin/gfortran-mp-4.4 \
  ./configure --prefix=$DGM_TPL/hdf5-1.8.9/serial/opt
% make install
% ./configure --prefix=$DGM_TPL/hdf5-1.8.9/mpi/opt --enable-parallel
```

or

```
% CC=/Users/ccober/local/openmpi/bin/mpicc \
  CXX=/Users/ccober/local/openmpi/bin/mpicxx \
  FC=/Users/ccober/local/openmpi/bin/mpif77 \
  ./configure --prefix=$DGM_TPL/hdf5-1.8.9/mpi/opt --enable-parallel
```

or if you get the following during configuring, `configure: error: unable to link a simple MPI-IO application`, try something like:

```
% CC=/Users/ccober/local/openmpi/bin/mpicc ./configure \
  --prefix=$DGM_TPL/hdf5-1.8.9/mpi/opt --enable-parallel
% make install
```

4) Create link and add HDF5 to your environment for ExodusII build.:

```
% cd $DGM_TPL ; ln -s hdf5-1.8.9 hdf5
% export HDF5_BASE_DIR=$DGM_TPL/hdf5/mpi/opt
```

5.2 NetCDF

ExodusII requires NetCDF version 4.2 or greater to build.

1) Download version 4.2 or later from http://www.unidata.ucar.edu/downloads/netcdf/netcdf-4_2/index.jsp (netcdf-4.2.tar.gz)

2) Move and uncompress NetCDF in build location (e.g., `$HOME/local` or `$DGM_TPL`):

```
% mv netcdf-4.2.tar.gz $DGM_TPL/.
% cd $DGM_TPL   # (e.g., $HOME/local)
% tar zxvf netcdf-4.2.tar.gz
```

3) Edit files per ExodusII README (Greg Sjaardema):

```
% cd netcdf-4.2
% vi include/netcdf.h
```

Make the following changes to these defines:

```
#define NC_MAX_DIMS 65536   /* max dimensions per file */
#define NC_MAX_VARS 524288  /* max variables per file */
#define NC_MAX_VAR_DIMS 8   /* max per variable dimensions */
```

and then make sure to make the stack size unlimited in your shell:

```
% ulimit -s unlimited       # To pass tests w/ new changes

or

% ulimit -Ss 65532
```

4) Configure and build serial and parallel netcdf. On the Mac you may have to disable doxygen (`--disable-doxygen`) in order to build successfully. On Redsky, you need to disable shared libraries (`--disable-shared`) or you can build HDF5 with shared library support enabled and set your runtime library path. Jenkins doesn't do this so it requires static libraries at this time

```
% export HDF5_BASE_DIR=$DGM_TPL/hdf5/serial/opt
% ./configure --prefix=$DGM_TPL/netcdf-4.2/serial/opt \
 --disable-shared \
 CXX=/usr/bin/g++ \
 CC=/usr/bin/gcc  \
 CPPFLAGS=-I$DGM_TPL/hdf5/serial/opt/include \
 LDFLAGS=-L$DGM_TPL/hdf5/serial/opt/lib
% make check install    # May fail but a 'make install' seems to still work.
```

and for an MPI enabled build:

```
% export HDF5_BASE_DIR=$DGM_TPL/hdf5/mpi/opt
% ./configure --prefix=$DGM_TPL/netcdf-4.2/mpi/opt \
  --disable-shared \
  CXX=mpicxx \
  CC=mpicc   \
  CPPFLAGS=-I$DGM_TPL/hdf5/mpi/opt/include \
  LDFLAGS=-L$DGM_TPL/hdf5/mpi/opt/lib
% make check install    # May fail but a 'make install' seems to still work.
```

5) Create link and add netcdf to your environment for ExodusII build

```
% cd $DGM_TPL
% ln -s netcdf-4.2 netcdf
% export NETCDF_DIR=$DGM_TPL/netcdf
```

5.3 ExodusII

This is ONLY needed if you do not build with SEACAS/Exodus from Trilinos. To build some of the utilities (e.g., `exo2ien` for converting Cubit meshes), one needs ExodusII built. If you haven't already done so, make sure that you have built and installed both HDF5 and NetCDF as described in the preceeding sections.

1) Download the latest version from http://sourceforge.net/projects/exodusii/

2) In the directory where you want to put the exodusii installation, execute:

```
% cd $DGM_TPL                      # (e.g., $HOME/local)
% tar zxvf exodusii-4.98.tar.gz
```

3) Configure exodusii:

```
% ln -s exodusii-4.98 exodusii
% cd exodusii

% cmake -DCMAKE_INSTALL_PREFIX=$DGM_TPL/exodusii \
```

```
-DNETCDF_INCLUDE_DIR=$NETCDF_DIR/include \
-DCMAKE_CXX_COMPILER=/usr/bin/g++ .
```

Do not forget to include the "." at the end of this command as it must be there.

4) Make, test and install:

```
% make check install
```

5.4 Trilinos

1) First off, set up your environment for DGM builds by getting a copy of Jenkins environment setup script, and modify it for your directory structure and machine:

```
% cp $DGM_DIST/regression/jenkins/dgm-env-setup-<machine>.bsh \
% $HOME/dgm-env-setup-<machine>.bsh
% # vi and set DGM_DIST, DGM_TPL, TRILINOS_DIST, NETCDF_DIR, ...
% source $HOME/dgm-env-setup-<machine>.bsh
```

2) Obtain a copy of the Trilinos source. Following notes are from a Trilinos email.

How to get Trilinos from github: Github allows both https and ssh access. Both are equally valid and both have their pros and cons in a Sandia environment. SSH is what we have been using so it may be more natural to continue with it, but the choice can be made on an individual basis

https: % git clone https://github.com/trilinos/Trilinos.git

Note that if you choose to use https you will need to make sure your proxies are set correctly on every machine that you intend to do work from. You can find the information for Sandia's proxies here:

https://sems.sandia.gov/qa/how-do-i-configure-sandia-proxy-settings

ssh: git@github.com:trilinos/Trilinos.git

Note that if you choose to use ssh you will need to upload your public key to github for each machine that you intend to do work from. Github has a convenient way to add keys to your profile through the website. You can find instructions on how to add keys at:

https://help.github.com/articles/generating-ssh-keys/#step-4-add-your-ssh-key-to-your-account

4) Change directories to a location where you want the Trilinos libraries to be built. A common location is directly in `$DGM_DIST/util` directory. The script ran in the next step will create (or use) the directory `trilinos_build` in this location:

```
% cd $DGM_DIST/util
```

5) Run the `$DGM_DIST/util/make_trilinos` script. You can no longer make and install several Trilinos builds at the same time. You have to separately build Trilinos for serial and parallel due to HDF5.:

```
% export TRILINOS_HOME=$DGM_TPL/trilinos

% # serial
% export NETCDF_BASE_DIR=$DGM_TPL/netcdf/serial/opt
% export HDF5_BASE_DIR=$DGM_TPL/hdf5/serial/opt
% ./make_trilinos -cisn 8 $TRILINOS_DIST
```

Use the following to rebuild from scratch:

```
% ./make_trilinos -CIsn 8 $TRILINOS_DIST
```

but do not use the "-CI" during the parallel build as it will remove the serial build.:

```
% # parallel
% export NETCDF_BASE_DIR=$DGM_TPL/netcdf/mpi/opt
% export HDF5_BASE_DIR=$DGM_TPL/hdf5/mpi/opt
% ./make_trilinos -cipn 8 $TRILINOS_DIST
```

If you want debug Trilinos builds, you'll need to build and install debug builds of both HDF5 and Netcdf before building Trilinos.

You may need to remove any previous build of Trilinos to build without errors.:

```
% rm -rf trilinos_build
```

This command no longer works but the option set is still valid:

```
% ./make_trilinos -ispydn 4 $TRILINOS_DIST
```

will make serial (`-s`) and parallel (`-p`) versions of optimized and debug (`-d`) builds of Trilinos with dynamic libraries and PyTrilinos enabled (`-y`), using 4 processors (`-n 4`) and then install (`-i`) the builds in `$HOME/local` with the following directories:

```
$HOME/local/trilinos/mpi/debug
$HOME/local/trilinos/mpi/opt
$HOME/local/trilinos/serial/debug
$HOME/local/trilinos/serial/opt
```

You can change the installation destination by setting the environment variable `TRILINOS_HOME`. You can also specify the location of the MPI compilers with environment variable `MPI_BASE_DIR` (if this is not set, `make_trilinos` will first look in `~/local/openmpi`, and if that is not present the script will let the Trilinos build system look for it).

Note: If the `-y` option is specified, then `make_trilinos` will check for the prerequisites for PyTrilinos (python, swig and numpy). If these are present, then PyTrilinos will be enabled. The TriKota package requires the user to copy the Dakota source into the `packages/TriKota` directory of the Trilinos source. If this has been done, `make_trilinos` will detect it and enable TriKota.

5.5 Metis

Metis is both a library and a collection of command-line executables that enables the partitioning of mesh and graph datastructures.

1) Obtain a copy from http://glaros.dtc.umn.edu/gkhome/fsroot/metis/OLD. These instructions are for version 4 e.g., `metis-4.0.3.tar.gz`. Things changed quite a bit in version 5, but I have found that version 5 yeilds higher-quality partitions and that the command-line interface is more powerful, so you may want to explore that version too.

2) In the directory where you want to put the metis installation, execute:

```
% cd $DGM_TPL   # (e.g., $HOME/local)
% tar zxvf metis-4.0.3.tar.gz
% cd metis-4.0.3
```

3) Follow the directions in INSTALL, e.g., change to use gcc compiler in `Makefile.in`:

```
% vi Makefile.in  (CC = gcc)
% make
```

4) Test the build:

```
% cd Graphs
% mtest 4elt.graph
```

5) Create link to build:

```
% cd $DGM_TPL
```

5.6 Flex

Flex should be available on your sytem and under most circumstances, you will not need the instructions that follow. However, for some (older) systems, the default flex is known to have a bug and must be patched. If you have this situation, please follow these steps.

1) Download the latest version of flex and patch. - From https://launchpad.net/ubuntu/+source/flex/2.5.35-10 - original and patch: flex_2.5.35.orig.tar.gz and flex_2.5.35-10.diff.gz

2) cd /to/the/source/code/location:

```
% cd $DGM_TPL   # (e.g., $HOME/local)
```

3) Move original to source location:

```
% mv flex_2.5.35.orig.tar.gz $DGM_TPL
% tar zxvf flex_2.5.35.orig.tar.gz
```

4) Move patch to flex directory:

```
% cp flex_2.5.35-10.diff.gz $DGM_TPL/flex-2.5.35
```

5) Apply patch:

```
% cd $DGM_TPL/flex-2.5.35
% gunzip flex_2.5.35-10.diff.gz
% patch -p1 < flex_2.5.35-10.diff
```

6) Configure and build:

```
% ln -sf $DGM_TPL/flex-2.5.35 $DGM_TPL/flex
% ./configure --prefix=$DGM_TPL/flex
% make
% make check      # (may fail a test but seems to still work)
% make install
```

If you get the following, `/bin/sh: $HOME/local/metis/install: Permission denied`, you may not have applied the patch.

5.7 JsonCpp

The JsonCpp library is now embedded within DGM so that you should not need to download and follow these instructions. However, we are keeping them here just in case.

1) The JsonCpp library can be found at http://jsoncpp.sourceforge.net/:

```
% cd $DGM_TPL
% tar zxvf jsoncpp-src-0.6.0-rc2.tar.gz
```

2) Can be built (please check the current READ.txt), e.g.:

```
% cd $DGM_TPL/jsoncpp-src-0.6.0-rc2
% python $DGM_TPL/scons-2.0.1/bin/scons platform=linux-gcc check
```

3) At the moment, we use 0.6.0-rc2. JsonCpp does not currently include an install script. As a result, the library is found at:

   ```
   jsoncpp-src-0.6.0-rc2/libs/{ARCH}/libjson_{ARCH}_libmt.a
   ```

   and the include files are found at:

   ```
   jsoncpp-src-0.6.0-rc2/include/json
   ```

   Since JsonCpp appends the architecture and compiler to the library name and directory, the default build REQUIRES a symbolic link in the:

   ```
   jsoncpp-src-0.6.0-rc2/libs
   ```

   directory to `libjson_{ARCH}.a` with the name `libjson.a`. For example, in the libs directory, the following would suffice on a Linux system:

   ```
   % cd $DGM_TPL/jsoncpp-src-0.6.0-rc2/libs
   % ln -s ./linux-gcc-4.1.2/libjson_linux-gcc-4.1.2_libmt.a ./libjson.a
   ```

4) Also to complete the build you will need to set an environment variable:

   ```
   % setenv USE_JSON 1
   ```

   and ensure that is in your path, e.g.:

   ```
   % ln -s $DGM_TPL/jsoncpp-src-0.6.0-rc2 $DGM_TPL/jsoncpp
   ```

5.8 SCons

The SCons tool is needed to help build JsonCpp. You can learn more about SCons at http://www.scons.org. Since JsonCpp is now built into DGM, this is really no longer needed but is retained just in case.

1) Untar:

   ```
   % cd $DGM_TPL
   % tar zxvf scons-2.0.1.tar.gz
   ```

2) Install:

   ```
   % cd $DGM_TPL/scons-2.0.1
   % python setup.py install --prefix=$DGM_TPL/scons-2.0.1
   ```

5.9 Shared Libraries

A common issue when building and running DGM is that some third-party libraries may default to using shared (or dynamic) libraries which are resolved at runtime. For example, under Linux you might need to set:

```
% export LD_LIBRARY_PATH=$DGM_TPL/gsl/lib:$LD_LIBRARY_PATH
```

in your `.bash_profile` file. In Darwin (Mac OS-X) you would set:

```
% export DYLD_LIBRARY_PATH=$DGM_TPL/gsl/lib:$DYLD_LIBRARY_PATH
```

6 Troubleshooting

6.1 Undefined symbols: "_dgm_yylex"

This is a problem that crops up with Leopard - fortunately fixed in Snow Leopard. The issue is that the version of flex that is distributed with Leopard was messed up. So, use a version that is built that is in:

```
$DGM_TPL/flex/bin/flex
```

by adding the following to your path:

```
export PATH=$DGM_TPL/flex/bin:$PATH
```

You may need to clean the build after adding these to your path.

6.2 Undefined reference to 's_wsfe' (libf77blas.a)

On Odin, you have probably built with GCC instead of ICC. Try:

```
% make_dgm -cisp -a ICC
```

6.3 Comparison signed and unsigned error in lex

If you see the error:

```
../src/lex.yy.cpp: In function `int yy_get_next_buffer()':
../src/lex.yy.cpp:1091: warning: comparison between signed and
unsigned integer expressions
```

You are using a version of flex which does not have the latest patch. Many system version do not have the patch. Check to see which flex you are using:

```
% which flex
  /Users/ccober/local/flex/bin/flex
```

and make sure it is your local version. You may need to rebuild flex with the latest patch (possibly `2.5.35-10` with `flex_2.5.35-10.diff`). See the Flex section above.

You may also need to remove some intermediate files, so will need to do a distribution clean, e.g.:

```
% make_dgm -CispT
```

6.4 Dereferencing pointer when building with peopt

The error looks something like:

```
/Users/ccober/local/peopt/include/peopt/peopt.h:4633: error: dereferencing
pointer '<anonymous>' does break strict-aliasing rules
/Users/ccober/local/peopt/include/peopt/peopt.h:4633: note: initialized
from here
```

This error occurs when using GCC 4.4 with peopt. This was a known and fixed GCC bug, which was documented here:

http://gcc.gnu.org/bugzilla/show_bug.cgi?id=39390

http://gcc.gnu.org/bugzilla/show_bug.cgi?id=42087

This has been fixed for GCC 4.5 and later versions. To get around it, try the following:

```
% env MCPU=-fno-strict-aliasing ./make_dgm -ispT
```

6.5 runtests.py fails; undefined reference

The full error is:

```
runtests.py fails; undefined reference to
DGM::Domain::reset_system_matrix()
```

This error indicates that the executable you are using was compiled without trilinos. So:

```
% make_dgm -CispT $TRILINOS_HOME
```

and then update your path to make sure that the executable with Trilinos in it is picked up first. Do:

```
%echo $PATH
```

and check which dgm.exe comes first. Check if `$DGM_HOME/dgm/trilinos/mpi/opt/bin/dgm.exe` (or `$DGM_HOME/dgm/trilinos/serial/opt/bin/dgm.exe`) is early in your path. `$DGM_HOME` should be set to something in your home directory and not to `/home/dgm/local`

6.6 No newline at end of file

The error:

```
/ascldap/users/ccober/local/boost/include/boost/fusion/tuple/detail/
preprocessed/tuple_tie.hpp:21:7: error: no newline at end of file
```

happens on redsky when the default architecture is used (GCC), but the TLCC architecture should be used, e.g., the following was simply used:

```
% make_dgm -isp
```

The following should fix this:

```
% make_dgm -ispa TLCC
```

6.7 OpenMPI hangs

Under MacOS X 10.9 (Mavericks), OpenMPI can hang in the MPI_Init() function if your computer is connected to the internet. See this thread for more information.

The best solution is to set the following environment variable:

```
% OMPI_MCA_oob_tcp_if_include=lo0
```

6.8 Illegal Instruction (Muzia)

If after building on muzia and during execution, you get the following execution error:

```
% time aprun -q -n 128 pdgm_opt.exe -no-rst cyl
    _pmiu_daemon(SIGCHLD): [NID 00015] [c0-0c0s7n1]
    [Fri Jun  6 11:56:01 2014] PE RANK 51 exit signal Illegal instruction
  ...
```

you probably have some conflicting compiler options. Goto config/CRAY.inc:line 243, and make sure OPTXX is set to:

```
OPTXX   = $(CXXFLAGS) -O2 $(MCPU)
```

Noel Belcourt has noted that additional compiler options other than -O2 can cause this illegal instruction.

7 Cray Instructions

These are details on the installation and building on Cray platforms. Follow the general installation instructions in Install.txt, except for those steps noted here.

1) Use someone else's TPL builds if possible:

```
% cd ${HOME}
% mkdir local
% cd local
% ln -s /home/sscoll/local/atlas .
% ln -s /home/sscoll/local/boost .
% ln -s /home/sscoll/local/cblas .
% ln -s /home/sscoll/local/cmake .
% ln -s /home/sscoll/local/git .
% ln -s /home/sscoll/local/peopt .
% ln -s /home/sscoll/local/trilinos .
```

2) Use Cray's build of HDF5 and NetCDF:

```
% mkdir hdf5
% cd hdf5
% mkdir serial mpi
% cd serial
% ln -s /opt/cray/hdf5/default/cray/83 opt
% cd ../mpi/
% ln -s /opt/cray/hdf5-parallel/default/cray/83 opt
% cd ../..
% mkdir netcdf
% cd netcdf
% mkdir serial mpi
% cd serial
% ln -s /opt/cray/netcdf/default/cray/83 opt
% cd ../mpi
% ln -s /opt/cray/netcdf-hdf5parallel/default/cray/83 opt
```

3) Checkout codes:

```
% cd ${HOME}
% svn co svn+ssh://software.sandia.gov/svn/private/dgm/trunk dgm
% svn co svn+ssh://dev.sandia.gov/usr/local/svn/peopt/trunk peopt
% ~/local/git/bin/git clone software.sandia.gov:/space/git/Trilinos
% ~/local/git/bin/git clone software.sandia.gov:/space/git/preCopyrightTrilinos
% cd Trilinos/
% ln -sf ../preCopyrightTrilinos .
```

4) Setup environment:

```
% cd ${HOME}
```

Edit these as needed:

```
% cp dgm/regression/jenkins/dgm-env-setup-curie.bsh .
```

5) Build Trilinos

Building Trilinos for a Cray XK-6, use the following:

```
% # serial
% module load cray-hdf5 cray-netcdf
% export NETCDF_DIR=/opt/cray/netcdf/default/cray/83/
% export NETCDF_BASE_DIR=/opt/cray/netcdf/default/cray/83/
% export HDF5_BASE_DIR=/opt/cray/hdf5/default/cray/83/
% make_trilinos -D CMAKE_C_COMPILER="cc" -D MPI_CXX_COMPILER="CC" \
  -D BLAS_LIBRARY_NAMES:STRING="" -D LAPACK_LIBRARY_NAMES:STRING="" \
  -D BUILD_SHARED_LIBS:BOOL=FALSE -D TPL_FIND_SHARED_LIBS:BOOL=FALSE \
  -D Trilinos_LINK_SEARCH_START_STATIC:BOOL=ON -isn 8 $TRILINOS_DIST

% # parallel
% module load cray-netcdf-hdf5parallel cray-hdf5-parallel
```

or

```
% module swap cray-netcdf cray-netcdf-hdf5parallel
% module swap cray-hdf5 cray-hdf5-parallel

% export NETCDF_DIR=/opt/cray/netcdf-hdf5parallel/default/cray/83/
% export NETCDF_BASE_DIR=/opt/cray/netcdf-hdf5parallel/default/cray/83/
% export HDF5_BASE_DIR=/opt/cray/hdf5-parallel/default/cray/83/
% make_trilinos -D CMAKE_C_COMPILER="cc" -D MPI_CXX_COMPILER="CC" \
  -D BLAS_LIBRARY_NAMES:STRING="" -D LAPACK_LIBRARY_NAMES:STRING="" \
  -D BUILD_SHARED_LIBS:BOOL=FALSE -D TPL_FIND_SHARED_LIBS:BOOL=FALSE \
  -D Trilinos_LINK_SEARCH_START_STATIC:BOOL=ON -ipn 8 $TRILINOS_DIST

% export NETCDF_DIR=/opt/cray/netcdf/default/cray/83/
% export NETCDF_BASE_DIR=/opt/cray/netcdf/default/cray/83/
% export HDF5_BASE_DIR=/opt/cray/hdf5/default/cray/83/
% module swap cray-netcdf-hdf5parallel cray-netcdf
% module swap cray-hdf5-parallel cray-hdf5
```

6) Build DGM:

```
% cd ${DGM_DIST}
% make_dgm -ispT -a CRAY
```

7) Build DGM utilities:

```
% cd ${DGM_DIST}/util
% make
```

8 ROMIO Setup

On the RedSky platform, it is necessary to configure options to the ROMIO system to get acceptable performance on the Lustre filesystem. *These changes are not necessary on Cray platforms.*

Create a `.romio-hints` file in your home directory and put the following into that file:

```
cb_nodes 1
romio_no_indep_rw true
```

Set your environment to point to the hints file:

```
export ROMIO_HINTS=$HOME/.romio-hints
```

9 Graph Weights Setup

In order for Zoltan to use the graph weights during parallel mesh decomposition, the graph weights database must be generated.

9.1 Building the weights database

1) Build serial DGM:
   - ensure `dgm.exe` is in your `PATH`
   - ensure `PYTHONPATH` can find `dgm_test_support.py`
2) Checkout and change to this test directory:

   ```
   % cd dgm/runs/graph_weights
   ```
3) Run the test, it may run for several hours so consider running the tests in the background:

   ```
   % nohup ./physics.tst &
   ```
4) When the graph weights test has completed, the database will be written to:

   ```
   $DGM_HOME/performance/dgm_weights.txt
   ```
5) In order for Zoltan to use the weights database, you must manually copy the file from:

   ```
   $DGM_HOME/performance/dgm_weights.txt
   ```

   to

   ```
   $DGM_HOME/trilinos/mpi/opt/bin
   ```

9.2 Using the weights database

A DGM test will use the weights database if:

- a Trilinos-enabled build of `pdgm.exe` is used
- The weights database resides in the bin directory or resides in the test directory. If the test Json input supplies a path to the weights database, that file will be used if it exists.
- The test.json file enables Zoltan and the graph weights:

```
"Zoltan" : {
  "Partition" : {
    "algorithm" : "hypergraph"
  },
  "Weights" : {
    "File" : "dgm_weights.txt"
  },
  "Write Partition" : true
}
```

- Zoltan will be used only if a binary mesh is specified in the input (`bmesh = 1`)
- The test is parallel (`np > 1`)
- No `root.part.np` file exists. If this partition file exists for the requested number of processors, this file will be used and Zoltan will not.