Overture installation instructions on a Ubuntu 14.04 (64-bit) from scratch

Author: Chahe Adourian

Date: 2014-June-16

Version: 0.2

Table of Contents

1. Scope

Overture installation procedure on a freshly installed Ubuntu Operating system.

1.1. Version description details

This installation procedure was tested on the following setup:

1. Operating system: Ubuntu 14.04 AMD64
2. Overture version: 25
3. CG version: 25
4. A++P++ version: 0.8.1
5. PETSC version: 2.3.2-p10

2. Prequisites

Before downloading and compiling the Overture files, there a packages that need to be downloaded and installed. This is easily done using apt-get command using a root account. The detailed commands are listed below.

2.1. Ubuntu Linux 14.04 LTS for AMD64 installation

This installation was tested on Ubuntu 14.04 LTS for AMD64. If you have a Ubunut version that is close enough, the procedure may work. If you wish to be on the safe side, you can download and install Ubuntu 14.04 amd64 (or a later version) before proceeding with the rest of this document.

Notes:

- if you are not using an 64bit OS, there would have to be a few changes to this procedure which were not investigated but they would have to do with paths in the defenv_bash file
  - export HDF=/usr/lib/x86_64-linux-gnu
- and the Overture package's ./configure file where the path to the LIB64 files were modified. Possibly, the path to the 32bit lib files will have to be modified as well.

2.2. Update Ubuntu packages and install Overture required packages

Execute the following linux commands

Linux command	**Description**
sudo apt-get update	Install package updates
sudo apt-get install build-essential manpages-dev gfortran autoconf automake mpich2 csh tcsh libmotif-dev libgl1-mesa-dev libglu1-mesa-dev x11proto-gl-dev x11proto-print-dev libzip-dev libperl-dev libXpm-dev libXp-dev libxmu-dev libxi-dev libhdf5-dev autoconf liblapack-dev libblas-dev	Install required packages before compiling Overture packages

3. Download and deploy Overture source code

This section described how to download, deploy, compile and install Overture and associated projects.

Important notes:

1. This procedure was tested with Overture version 25. Certain adjustments are required to the Overture source code which are needed because the compiler requirements have changed and code that was error-free now raises errors and warnings.
2. Certain errors were found in the Makefiles and perl scripts that ran tests. Instructions for correcting these are provided in this section.

3.1. Acquire and deploy source code

Download and deploy all the Overture files using the following linux commands

Linux command	**Description**
mkdir ~/Overture-Install	Make the installation directory in the user's home directory
cd ~/Overture-Install	Go to the download and compilation directory
wget http://overtureframework.org/software/Overture.v25.tar.gz	Download overture source code from Overture website
wget http://overtureframework.org/software/cg.v25.tar.gz	Download CG source code from Overture website
wget http://www.overtureframework.org/software/AP-0.8.1.tar.gz	Download A++P++ source code from Overture website
wget http://ftp.mcs.anl.gov/pub/petsc/software_old/petsc-2.3.2.tar.gz	Download PETSC source code
tar zxvf Overture*.tar.gz	Uncompress Overture source code
tar zxvf cg*.tar.gz	Uncompress cg source code
tar zxvf AP*.tar.gz	Uncompress A++P++ source code
tar zxvf petsc*.tar.gz	Uncompress PETSC source code

3.2. Compilation environment settings file

Before compiling Overture and CG, environment variables need to be set. A bash script file needs to be created. Here are the steps:

1. Create file: touch ~/Overture-Install/defenv_bash
2. Edit the file and enter the following text

Contents of file: defenv_bash

```
#! /bin/bash
#
# Define environment variables for Overture
#
# see the install notes on the web for further explanation or Overture/doc/install.tex
#
# edit this file and change as appropriate for your system,
# then type 'source defenv_bash'

#### USER CONFIGURED PARAMETERS ========================
# Configure where the Overture files will be decompressed and compiled from
export OVERTUREINSTALL=$HOME/Overture-Install

# Folders where the Package files are decompressed
export OVERTURE_DIR=$OVERTUREINSTALL/Overture.v25
export PETSC_DIR=$OVERTUREINSTALL/petsc-2.3.2-p10
export APLUSPLUS_DIR=$OVERTUREINSTALL/A++P++-0.8.1
export CG_DIR=$OVERTUREINSTALL/cg.v25

# Select the architecture for which the PETSC code will be compiled. For a list of options,
# see the python filenames in the petsc-*/config. The architectures are in the prefix of the python files
# See also the petsc-*/docs/installiation.html for instructions
export PETSC_ARCH=linux-gnu-amd64

#### END OF USER CONFIGURED PARAMETERS ========================
export Overture=$OVERTURE_DIR
export APlusPlus=$APLUSPLUS_DIR/A++/install

#export MPIPATH=/opt/anaconda     # Works - uses mpi.h

export XLIBS=/usr
export MOTIF=/usr
export OpenGL=/usr

# Location of the HDF5 libraries
export HDF=/usr/lib/`uname -m`-linux-gnu
#export HDF=/usr/lib/x86_64-linux-gnu

# Here are variables to set if you intend to build CG
#   CG : location for CG (source tree
#   CGBUILDPREFIX : location of CG executables (compile tree). This can be the same as CG
#                   if you only intend to build one version of CG.

export CG=$CG_DIR
```

```
export CGBUILDPREFIX=$CG_DIR

#   LAPACK : location of the LAPACK library
#  In the following example LAPACK is found with the PGI compilers
export LAPACK=/usr/lib/lapack

# Here are some handy alias's and variables
alias ogen=$OVERTURE_DIR/bin/ogen
alias plotStuff=$OVERTURE_DIR/bin/plotStuff
alias rap=$OVERTURE_DIR/bin/rap
alias mbuilder=$OVERTURE_DIR/bin/mbuilder
export primer=$OVERTURE_DIR/primer
export sampleGrids=$OVERTURE_DIR/sampleGrids

export CGE=$CGBUILDPREFIX
alias cgad=${CGE}/ad/bin/cgad
alias cgasf=${CGE}/asf/bin/cgasf
alias cgcns=${CGE}/cns/bin/cgcns
alias cgins=${CGE}/ins/bin/cgins
alias cgmx=${CGE}/mx/bin/cgmx
alias cgmp=${CGE}/mp/bin/cgmp
alias cgsm=${CGE}/sm/bin/cgsm

export ad=${CG}/ad
export asf=${CG}/asf
export cns=${CG}/cns
export ins=${CG}/ins
export mx=${CG}/mx
export mp=${CG}/mp
export sm=${CG}/sm

# The LD_LIBRARY_PATH indicates the places to look for dynamic libraries. This variable should
# not be needed in most cases since we use the 'rpath' loader option
export LD_LIBRARY_PATH=$MOTIF/lib:$XLIBS/lib:$OpenGL/lib:$HDF/lib:
$OVERTURE_DIR/lib:${APlusPlus}/lib
#export LD_LIBRARY_PATH=$MOTIF/lib:$XLIBS/lib:$OpenGL/lib:$HDF/lib:
$OVERTURE_DIR/lib:${APlusPlus}/lib:${PETSC_LIB}
```

3.3. Bug fixes

Before proceeding with the compilation, there are changes to be made to the Overture and CG source codes, compilation and test scripts. Some of the changes fix bugs and some are adaptations to the code to work in the newer Ubuntu 14 Operating system since the Overture v.25 files were working on Ubuntu 10 and its associated compilers.

3.3.1. Overture v25 bug fixes

The following are bug fixes to be made to the Overture project files before compilation is started.

3.3.1.1.Bug fix: Overture.v25/configure

Edit file ~/Overture-Install/Overture.v25/configure
replace $LIB64="lib64"; ==> $LIB64="lib/x86_64-linux-gnu";

This is the original text:

```
# On some 64 bit machines some of the libraries are called "lib64" instead of "lib"
#     x86_64 : intel 64 bit machine
$machineType = `uname -m`;
chop($machineType);
printf(" *** machineType = [$machineType] **** \n");
$LIB64="lib";
if( $machineType eq "x86_64" )
{
  $LIB64="lib64";
}
```

This is the modified text:

```
# On some 64 bit machines some of the libraries are called "lib64" instead of "lib"
#     x86_64 : intel 64 bit machine
$machineType = `uname -m`;
chop($machineType);
printf(" *** machineType = [$machineType] **** \n");
$LIB64="lib";
if( $machineType eq "x86_64" )
{
#  $LIB64="lib64";
   $LIB64="lib/x86_64-linux-gnu";
}
```

3.3.1.2.Bug fix: Overture.v25/lib/Makefile.in

Edit file ~/Overture-Install/Overture.v25/lib/Makefile.in
add string “-lgfortran” to the make rule *libso_date*
The addition is seen in yellow text.
This is the new text.

```
libso_date: $(SO_LIB_DEPENDENCIES) $(DataBase)/DataBase_date \
        $(GridFunction)/GridFunction_date          \
        $(Mapping)/Mapping_date            \
```

```
        $(Oges)/Oges_date               \
        $(Grid)/Grid_date         \
        $(GridGenerator)/GridGenerator_date \
        $(Ogshow)/Ogshow_date  \
           $(otherStuff)/otherStuff_date \
        $(templates)/ListClasses_date \
           $(mapUtil)/mapUtil_date
           $(LD) -o $(DYNAMIC_LIBRARY) $(SOFLAGS) $(DataBase)/*.o $(GridFunction)/*.o $
(templates)/*.o \
          $(Grid)/*.o $(Mapping)/*.o $(Oges)/*.o  $(GridGenerator)/*.o $(Ogshow)/*.o $(mapUtil)/*.o
\
             $(otherStuff)/*.o $(LIBLIBS) -lgfortran
           rm -rf libOverture.so.1.0
           ln -s $(DYNAMIC_LIBRARY) libOverture.so.1.0
           touch $@
```

3.3.2. Bug fixes to Overture test scripts

Running test scripts on the Overture folder generated multiple errors, here are some of the solutions we found.

3.3.2.1.Bug fix file: Overture.v25/sampleGrids/square.cmd

Edit Overture.v25/sampleGrids/square.cmd
change line 14, add closing braces "}" at the end of line.

Initial code

```
if ( $saveGrid ) { $finish = "save an overlapping grid\n$name\nsquare\nexit";
```

Fixed code

```
if ( $saveGrid ) { $finish = "save an overlapping grid\n$name\nsquare\nexit";}
```

3.3.3. CG v25 bug fixes

The following are bug fixes to be made to the CG project files before compilation is started.

3.3.3.1.Bug fix file: cg.v25/common/src/userDefinedKnownSolution.C

Edit File: CG.v25/common/src/userDefinedKnownSolution.C

Insert the "(const *char)" typecases

replace

```
    printf("getUserDefinedKnownSolution:ERROR: unknown value for
userDefinedKnownSolution=%i\n",
       userKnownSolution);

with

    printf("getUserDefinedKnownSolution:ERROR: unknown value for
userDefinedKnownSolution=%i\n",
       (const char*)userKnownSolution);
```

3.3.3.2.Bug fix file: cg.v25/common/src/errors.C

Edit file and replace certain references to variable “norm” by “inorm” as detailed below.

This is the initial Code

```
  for( int norm=0; norm<numberOfNormsToPrint; norm++ )
  { // norm==0 : max-norm, otherwise Lp-norm with p=norm
    int pNorm = norm==0 ? INT_MAX : norm;
    err=0.;
    for( int n=0; n<parameters.dbase.get<int >("numberOfComponents"); n++ )
    {
        if( pNorm<10000 )
        {
          err(n)=lpNorm(pNorm,v,n,maskOption,parameters.dbase.get<int
>("checkErrorsAtGhostPoints") );
        }
        else
        { // assume this is the max-norm
          err(n)=maxNorm(v,n,maskOption,parameters.dbase.get<int
>("checkErrorsAtGhostPoints") );
        }
    }
```

This is the new code after variable name “*norm*” was replaced with “*inorm*”

```
  for( int inorm=0; inorm<numberOfNormsToPrint; inorm++ )
  { // norm==0 : max-norm, otherwise Lp-norm with p=norm
    int pNorm = inorm==0 ? INT_MAX : inorm;
    err=0.;
    for( int n=0; n<parameters.dbase.get<int >("numberOfComponents"); n++ )
    {
        if( pNorm<10000 )
        {
          err(n)=lpNorm(pNorm,v,n,maskOption,parameters.dbase.get<int
```

```
>("checkErrorsAtGhostPoints") );
        }
        else
        { // assume this is the max-norm
          err(n)=maxNorm(v,n,maskOption,parameters.dbase.get<int
>("checkErrorsAtGhostPoints") );
        }
    }
```

3.3.3.3.Bug fix file: cg.v25/asf/src/asfp.C

Many printf statements no longer compile because the compiler is more restrictive. There are two solutions one of which is to comment out those lines. This is the one described here. A better option would be to make the necessary changes to make the printf codes work again.

Here are all the changes to be made to the source code.

Edit file: cg.v25/asf/src/asfp.C

On line 851, comment out the printf code

```
printf("cg[grid=%i].boundaryCondition()(side=%i,axis=%i)=%i (%s)\n",grid,side,axis,
                cg[grid].boundaryCondition()(side,axis),
                parameters.bcNames[cg[grid].boundaryCondition()(side,axis)]);
```

On line 1337:, comment out the printf code

```
printf("cg[grid=%i].boundaryCondition()(side=%i,axis=%i)=%i (%s)\n",grid,side,axis,
                cg[grid].boundaryCondition()(side,axis),
                parameters.bcNames[cg[grid].boundaryCondition()(side,axis)]);
```

On line: 1517, comment out printf code

```
printf("cg[grid=%i].boundaryCondition()(side=%i,axis=%i)=%i (%s)\n",grid,side,axis,
                cg[grid].boundaryCondition()(side,axis),
                parameters.bcNames[cg[grid].boundaryCondition()(side,axis)]);
```

3.3.3.4.Bug fix file: cg.v25/asf/src/formAllSpeedPressureEquation.C

Similarly, comment out problematic printf codes or better still fix the problem.

On line 478, comment out printf code

```
printf("cg[grid=%i].boundaryCondition()(side=%i,axis=%i)=%i (%s)\n",grid,side,axis,
        cg[grid].boundaryCondition()(side,axis),
        parameters.bcNames[cg[grid].boundaryCondition()(side,axis)]);
```

3.3.3.5.Bug fix file: cg.v25/mx/src/getErrors.C

Edit file ~/Overture-Install/cg.v25/mx/src/getErrors.c

Rename norm to inorm in this code on line 2085

Initial code

```
for( int norm=0; norm<numberOfNormsToPrint; norm++ )
{ // norm==0 : max-norm, otherwise Lp-norm with p=norm
   int pNorm = norm==0 ? INT_MAX : norm;

   if( norm!=0 ) // max-norm values are already computed -- we could avoid doing this above --
```

Modified code

```
for( int inorm=0; inorm<numberOfNormsToPrint; inorm++ )
{ // norm==0 : max-norm, otherwise Lp-norm with p=norm
   int pNorm = inorm==0 ? INT_MAX : inorm;

   if( inorm!=0 ) // max-norm values are already computed -- we could avoid doing this above --
```

3.3.3.6.Bug Fix file: cg.v25/sm/src/getRayleighSpeed.C

Edit file ~/Overture-Install/cg.v25/sm/src/getRayleighSpeed.C

Change the printf to cout and make the necessary changes for that. Otherwise it will fail to compile.

Details: TBD

3.3.4. Bug fixes to CG test scripts

Running “make check” on the cg folder generated multiple errors, here are some of the solutions we found.

3.3.4.1.Bug fix file: cg.v25/cns/cmd/tz.cmd

There is a mistake in the command file cg.v25/cns/cmd/tz.cmd
Change the ":" to a ";" on line 42

Initial code

```
$axisym=0; $bc1="": $bc2=""; $bc3=""; $bc4="";  $bc5="";  $bc6="";
```

Modified code

```
$axisym=0; $bc1="; $bc2=""; $bc3=""; $bc4="";  $bc5="";  $bc6="";
```

3.3.4.2.Bug fix file: cg.v25/mp/cmd/twoDomain.cmd

On line 128, the variable ad21 should be $ad21 (perl variable).

Initial code

```
$ad2=0; ad21=0; $ad22=0.; $adcBoussinesq=0.;
```

```
```

Modified code

```
$ad2=0; $ad21=0; $ad22=0.; $adcBoussinesq=0.;
```

3.4. Compilation and tests

After the source code, test scripts and other files have been corrected as described above, we can proceed with the compilation and testing of the Overture packages.

3.4.1. Compiling Installing A++

From a bash shell, run the following commands

cd /usr/bin
sudo ln -s make gmake

source ~/Overture-Install/defenv_bash
cd $APLUSPLUS_DIR
./configure --enable-SHARED_LIBS --prefix=`pwd`
#compile
make
make install

 # Run tests
make check

3.4.2. Compiling and installing PETSC (Optional)

Detailed compilation instructions are found in file: *$PETSC_DIR*/docs/installation.html

The version downloaded was petsc-2.3.2-p10. Here are quick instructions to compile petsc

From a bash shell, run the following commands

source ~/Overture-Install/defenv_bash
cd $PETSC_DIR

./config/configure.py -with-debugging=0 -with-fortran=0 -with-matlab=0 -with-mpi=1 –with-shared=1 -with-dynamic=1

 # Compile and test the code
make all test

3.4.3. Compiling and install Mesa

Overture requires the Mesa 3D libraries for, among other things, off-screen rendering. Here are instructions to install MesaLib.

Note:
- the version of MesaLib is specifically selected not to be 10.2.x as this source code requires versions of other packages not available yet in Ubuntu 14.04 LTS. For now, we stick to version 10.1.5

From a bash shell, run the following commands

cd ~/Overture-Install
wget http://xorg.freedesktop.org/releases/individual/proto/dri3proto-1.0.tar.gz
wget ftp://ftp.freedesktop.org/pub/mesa/10.1.5/MesaLib-10.1.5.tar.gz
wget ftp://ftp.freedesktop.org/pub/mesa/demos/8.1.0/mesa-demos-8.1.0.tar.gz
wget ftp://ftp.freedesktop.org/pub/mesa/glut/MesaGLUT-7.9.2.tar.gz
wget http://cgit.freedesktop.org/xorg/util/macros/snapshot/util-macros-1.19.0.tar.gz

tar zxvf dri3proto-1.0.tar.gz
tar zxvf MesaLib.tar.gz*
tar zxvf mesa-demos-.tar.gz*
tar zxvf MesaGLUT.tar.gz*
tar zxvf util-macros.tar.gz*

sudo apt-get install flex bison libtool xorg-dev libdrm-dev libxmu-dev libxdamage-dev git expat llvm-dev xutils-dev libudev-dev libglew-dev libglewmx-dev

 # Install dri3proto
*cd dri3proto**
./configure
sudo make install
cd ..

 # Compile MesaLib
*cd Mesa-**
./configure
gmake
sudo make install
cd ..

#--------------------------
 # Optional packages below
#--------------------------
 # We added util-macros to solve error "/libGL.so: undefined reference to `_glapi_tls_Dispatch"
 # when compiling mesa-demos.
*cd util-macros**
./autogen.sh
./configure
sudo make install

cd ..

#Compile mesa-demos
*cd mesa-demos**
./configure
make
sudo make install
cd ..

3.4.4. Compile and test Overture

3.4.4.1. Option 1: new instructions

From a bash shell, run the following commands

source ~/Overture-Install/defenv_bash
cd ${OVERTURE_DIR}

 # Configure compiler settings
./configure

#build Overture
make

 # Run tests
export PATH=./:$PATH
./check.p

3.4.4.2. Option 2: original instructions

Follow the instruction on overture website (click here).

3.4.5. Compile CG

3.4.5.1. Option 1: new instructions

From a bash shell, run the following commands

source ~/Overture-Install/defenv_bash
cd ${CG}

#build CG (mx module not included)
make

 # By default the Maxwell solver is not built. Go to the `$CG/mx` *directory and type* `make` *if you want to*
 # use it.
cd mx

make

 # Here are some tests you can run
cd ..
export PATH=./:$PATH
make check

3.4.5.2. Option 2: original instructions

Follow the instruction on overture website to compile CG (click here).

3.5. Installation instructions

After the packages are compiled, the execution of overture applications should become part of the normal operating environment. Here is one approach towards this:

Assuming the user uses the bash shell, add the following commands to the user's bash startup scripts.

Edit the file: ~/.bashrc
Insert the following command: source ~/Overture-Install/defenv_bash

Now, after starting a new bash script, all the aliased commands specified in ~/Overture-Install/defenv_bash will be accessible from the bash shell.

4. Improvement ideas

The documentation above is limited in many ways. Here we list some of the improvements to add to this document.

1. Examine original Overture compilation and installation instructions and add any missing features/functionalities to this documentation
2. Test installation procedure on various operating systems and submit fixes and improvements to compilation scripts, test scripts and code.
3. Document compilation instructions for multi-processor machines.
4. Document other compilation options.
5. Create Overture debian packages for both source code and binaries.