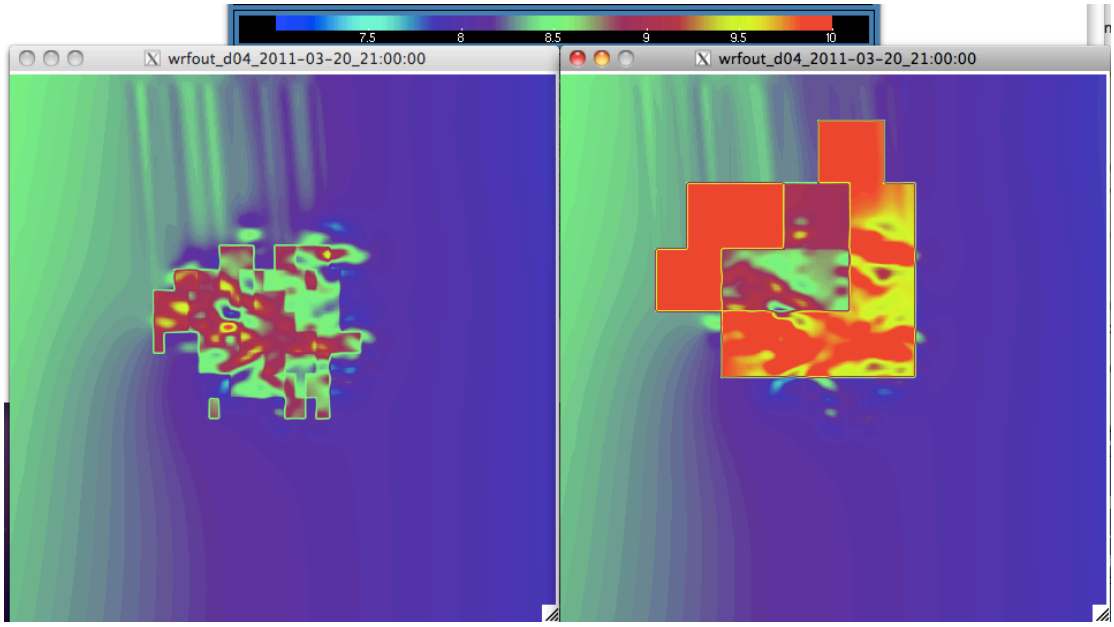
A system for high-resolution limited area numerical simulations

Technical Report

Work done for Uni BCCS



Simulated sfc wind speed over Utsira using Corine (left) and Modis (right) land use data

Institute for Meteorological Research
June 2011
Reykjavík, Iceland

Ólafur Rögnvaldsson
Þór Sigurðsson
Örnólfur E. Rögnvaldsson

✉ or@belgingur.is

✆ +354 528 1690

Introduction

A novel, open source, software solution has been developed that greatly simplifies running the WRF atmospheric model in Large Eddy Simulation (LES) mode. The work was funded by the University of Bergen, Norway, and carried out by the Institute for Meteorological Research in Iceland.

Running the WRF model in LES mode can be both time consuming and confusing. Typically, one runs the model with a regular planetary boundary layer (PBL) scheme down to a horizontal resolution of few kilometres using the model's nesting option. Using the output data from the innermost (i.e. highest resolution) PBL domain, one can create initial and boundary data for the WRF model in LES mode. This is done by using a component of the WRF modelling suite called `NDOWN` (short for Nest DOWN). Once that is done, one can finally run the WRF model in LES mode for the chosen area. Care must be taken when editing the WRF model's control files (called `namelist.wps` and `namelist.input`) during this procedure. As the user defines the domain setup "top down", it can be very time-consuming getting the exact location of the innermost domain correct. The new software package solves this by allowing the user to define the exact location, and extent, of the innermost domain. This is done either by defining two corner points of the domain, or by defining the centre (latitude and longitude) and radius of the domain. From this information the system then sets up the necessary control files in such a way that the innermost domain is as the user has defined. The system further sets up a unique directory structure for each simulation, copies or links relevant input data and creates the necessary runtime scripts that make it straight forward to run the WRF model through all the necessary steps.

Additionally, the system includes methods to use the near global 1 second ASTER topography data and the high resolution Corine land use data, that are available for large parts of Europe.

The system has been tested for a number of locations in Norway, Denmark and Iceland. Main usability is foreseen in research of wind energy for regions with relatively complex terrain.

System description and use

The WRFLES tool takes care of the following tasks:

- Setting up a directory hierarchy suitable for executing the steps of a WRF run.
- Calculating the namelist parameters corresponding to user defined criteria for the simulation region and generating the necessary namelists.
- Generating PBS scripts from templates that take care of the various steps in a WRF run.

The software is available on Hexagon under the directory `/home/geofysisk/olafurr/WRFLES`. The system is written in python and

consists of a number of modules contained in the src subdirectory. In addition, a number of pre-defined setups for specific regions are available under the systems subdirectory. In order to use the system, the user has to load the python-cnl/2.6.5-dynamic-exper module, using the command:

```
module load python-cnl/2.6.5-dynamic-exper
```

(It is possible that the system works with other versions of python, but it has only been tested with this version.)

The user interface consists of the following command line options and arguments:

```
olafurr@nid00163:~/WRFLES/src> python2.6 WRFLESSetup.py  -h
Usage: WRFLESSetup.py [options]

Options:
 -h, --help            show this help message and exit
 -d FILENAME, --default=FILENAME
                       Path to default configuration file (in YaML format)
 -s FILENAME, --specific=FILENAME
                       Path to specific configuration file (in YaML format)
 -o PATH, --output=PATH
                       Path to top level of output hierarchy (will be
                       created/emptied)
 -p YaML, --parameters=YaML
                       YaML encoded string specifying additional parameters
 -c, --clobber         Flag indicating whether to remove existing directory
                       tree
```

With the --default switch, the user specifies which configuration file to read. The parameters that control the generation of namelists and scripts are contained in this file. The --output switch argument defines the output directory where all subdirectories and files will be created.
For normal usage, the --parameters and --specific are not needed. Finally, the --clobber option is used to remove an output directory prior to creating it anew, so that the user can overwrite a previous setup.

To run the system the user should create a working directory under his/her home directory. For demonstration purposes we call this directory $HOME/wrf-les. There are three pre-defined setups available. One for the Havsul region, one for the island of Utsira and one for Bolund in Denmark. To edit the Bolund region, the user copies /home/geofysisk/olafurr/WRFLES/systems/bolund to $HOME/wrf-les/. Within this directory the user should only need to modify one control file, called WRFLES.config.yml:

```
# encoding: utf-8
# $Id: WRFLES.config.yml 2776 2011-04-08 00:24:52Z ossi $
#
# Test configuration file for WRF-LES
#
# Model parameters, used while generating namelist files
parameters:
  # The coordinates can either be specified as from_* and to_* or as
  # center_lat, center_lon, radius
  coordinates      :
    from_lat: 55.69765
    from_lon: 12.081275
    to_lat: 55.70906
    to_lon: 12.11015
```

```
  # or use center and radius (in units of km)
  # center_lat:
  # center_lon:
  # radius:
 # Specify start and end date for run
 start_date       : '2008-01-01_00:00:00'
 end_date         : '2008-01-02_00:00:00'
 # Specify the cutoffs from start and end for each nested domain
 # Take special care, as cutoff #1 has to be equal to cutoff #2 !!!
 start_cutoff     : [ '03:00:00', '03:00:00', '06:00:00', '06:00:00', '06:00:00' ]
 end_cutoff       : []
 # Specify the timestep in the input data
 input_data_timestep: '03:00:00'
 # The number of levels is fixed at 6, might want to remove the parameter from the config file
 number_of_levels   : 6
 # The resolution of the innermost domain in meters. Again, user should normally not change this
 resolution       : 50
 # The grid ratio for nested domains, same ordering as in namelist files
 grid_ratio       : [3, 2, 3, 3, 3]
 # The border width, number of points to add on each side of nested domains. The ordering
 # of this list is from the innermost domain and up
 # FIXME: We can change ordering so that it is same as for grid_ratio if this is
 # confusing for users.
 border_width     : [25, 20, 20, 40, 20, 17]
#
# Copy section describes how to populate directory tree with files
# A full wrf+wps setup will contain more files than included in the test
copy:
# namelist files
 - path: ./
   files:
     - '*namelist*'
     - GEOGRID.TBL
   destination: config
 - path: /home/geofysisk/olafurr/WRFLES/src
   files:
     - ModifyGeoem.py
     - ModifyMetem.py
   destination: bin
#
# Generation of PBS scripts
scripts:
 # The path to script templates
 path: ./scripts
 # Script will be prefix+key+postfix
 prefix: ''
 postfix: '.sh'
 # The subdir where scripts will be generated
 destination: bin
 # Common parameters and default values
 common_parameters:
   # Directories
   # Where is the system going to be run? The WRFLESSetup code
   # sets jobdir automagically to the output path specified on
   # command line. It is however possible to override it here
   #jobdir: /work/${USER}/bolund
   # Copy data from this directory to $jobdir/geogrid/geog/topo_1scustom
   asterdir: /home/geofysisk/olafurr/wrf/WPSV3.2/geog/topo_1sec-Bolund
   # Link the subdirs under this directory to same-name subdirs under
   # $jobdir/geogrid/geog/. geog_data_path is set to ./geog in namelist.wps
   geogdir: /work/shared/uib/mjo003/GEOG
   appswrfdir: /work/apps/WRF/3.2.1-pgi
   appswpsdir: /work/apps/WPS/3.2.1-pgi
   # Modules
   wpsmodule: WPS-cnl/3.2.1
   wrfmodule: WRF-cnl/3.2.1
   # Parameters for PBS
   project: geofysisk
   email: name@whatever..org
   email_flags: abe
   # Default expected resource consumption
   mppwidth: 16
   walltime: '00:30:00'
   mppmem: 1000mb
 # The remaining keys describe the scripts to generate
```

```
wrfles:
 jobname: wrfles-name
 mppwidth: 32
ndown:
 jobname: ndown-name
 walltime: '00:20:00'
 mppwidth: 16
preprocessing:
 jobname: preprocess-name
 mppwidth: 16
 mppnppn: 2
 mppmem: 2000mb
```

The parameters the user is most likely to want to change are marked in red and will be described in more detail:

- from_lat, from_lon, to_lat, to_lon – The innermost domain can be defined by setting a pair of corner points. The domain can also be defined by setting the center latitude and longitude, as well as radius, of the innermost domain. In the latter case the user comments out the from_* and to_* parameters and defines instead the center_lat, center_lon and radius parameters
- start_date – Initial time of the simulation
- end_date – Ending time of the simulation
- input_data_timestep – The time interval (hours) between available input data
- asterdir – The location of the 1 sec ASTER topography data for the region in question. Currently there is data available for three regions; Havsul, Utsira, and Bolund. This data is located at:
    - /home/geofysisk/olafurr/wrf/WPSV3.2/geog/topo_1sec-Havsul
    - /home/geofysisk/olafurr/wrf/WPSV3.2/geog/topo_1sec-Utsira
    - /home/geofysisk/olafurr/wrf/WPSV3.2/geog/topo_1sec-Bolund

  respectively.
- email – The e-mail the PBS queuing system should send information on "abe".

Once the user is happy with his/her modifications, the system is initiated by running the python script WRFLESSetup.py:

- python2.6 ~olafurr/WRFLES/src/WRFLESSetup.py -d $HOME/wrf-les/bolund/WRFLES.config.yml -o /work/$USER/bolund <enter>

This creates the following directory structure under /work/$USER/bolund (example taken from user olafurr):

```
olafurr@nid00008:/work/users/olafurr/bolund> ls -gF
total 36
drwxr-xr-x 2 olafurr 4096 2011-04-18 16:06 bin/
drwxr-xr-x 4 olafurr 4096 2011-04-12 18:00 config/
drwxr-xr-x 3 olafurr 4096 2011-04-12 18:32 geogrid/
drwxr-xr-x 2 olafurr 4096 2011-04-12 18:38 metgrid/
drwxr-xr-x 2 olafurr 4096 2011-04-12 01:21 ndown/
drwxr-xr-x 2 olafurr 4096 2011-04-12 18:38 real/
drwxr-xr-x 2 olafurr 4096 2011-04-09 23:22 ungrib/
drwxr-xr-x 2 olafurr 4096 2011-04-12 04:08 wrfles/
drwxr-xr-x 2 olafurr 4096 2011-04-09 23:35 wrfpbl/
```

There are three scripts under the bin directory and two python programs:

```
olafurr@nid00008:/work/users/olafurr/bolund/bin> ls -gF
```

```
total 20
-rw-r--r-- 1 olafurr 1727 2011-04-12 15:19 ModifyGeoem.py
-rw-r--r-- 1 olafurr 1727 2011-04-12 15:19 ModifyMetem.py
-rw-r--r-- 1 olafurr 1839 2011-04-09 23:22 ndown.sh
-rw-r--r-- 1 olafurr 4124 2011-04-12 17:27 preprocessing.sh
-rw-r--r-- 1 olafurr 2074 2011-04-12 03:51 wrfles.sh
```

The user runs these scripts via `qsub` one after the other, first `preprocessing.sh`, then `ndown.sh`, and finally `wrfles.sh`. However, before this can be done the user needs to copy, or link, the relevant intermediate files to the `ungrib` directory. All necessary configuration files are stored under the `config` directory. Those are:

```
olafurr@nid00008:/work/users/olafurr/bolund/config> ls -gF
total 120
-rw-r--r-- 1 olafurr 12413 2011-04-12 18:31 GEOGRID.TBL
drwxr-xr-x 2 olafurr  4096 2011-04-09 23:22 metgrid/
-rw-r--r-- 1 olafurr  2177 2011-04-01 14:18 namelist_data.input.ndown.yml
-rw-r--r-- 1 olafurr  7412 2011-04-05 17:30 namelist_data.input.real.yml
-rw-r--r-- 1 olafurr  2392 2011-04-07 19:58 namelist_data.input.wrfles.yml
-rw-r--r-- 1 olafurr   340 2011-04-01 14:18 namelist_data.input.wrfpbl.yml
-rw-r--r-- 1 olafurr  1494 2011-04-01 14:18 namelist_data.wps.yml
-rw-r--r-- 1 olafurr  5859 2011-04-09 23:22 namelist.input.ndown
-rw-r--r-- 1 olafurr  4505 2011-04-09 23:22 namelist.input.ndown.yml
-rw-r--r-- 1 olafurr  5948 2011-04-09 23:22 namelist.input.real
-rw-r--r-- 1 olafurr  4589 2011-04-09 23:22 namelist.input.real.yml
-rw-r--r-- 1 olafurr  5806 2011-04-09 23:22 namelist.input.wrfles
-rw-r--r-- 1 olafurr  4430 2011-04-09 23:22 namelist.input.wrfles.yml
-rw-r--r-- 1 olafurr  5950 2011-04-09 23:22 namelist.input.wrfpbl
-rw-r--r-- 1 olafurr  4591 2011-04-09 23:22 namelist.input.wrfpbl.yml
-rw-r--r-- 1 olafurr  1896 2011-04-09 23:22 namelist.wps
-rw-r--r-- 1 olafurr  1451 2011-04-09 23:22 namelist.wps.yml
drwxr-xr-x 2 olafurr  4096 2011-04-09 23:22 wrf/
```

The linking is taken care of in the three `.sh` scripts. By default, the last step, i.e. the `qsub wrfles.sh`, is only run for 30 minutes on 32 cores. This is mainly to ensure that the setup is stable and to allow the user to get an estimate of how long the actual simulation will take. Once that has been established, the user should edit the `wrfles.sh` PBS script and increase the simulation hours needed (`walltime:`), as well as the requested number of cores (`mppwidth:`).

Requesting ASTER data

The 1 sec ASTER topography data are free of charge and can be downloaded from the internet via the NASA "Warehouse Inventory Search Tool" (WIST) webpage:

- https://wist.echo.nasa.gov/api/

This webpage requires the user to register before being able to download any data. Detailed instructions can be found on the WIST webpage:

- https://wist.echo.nasa.gov/~wist/api/Tutorial/registered_user.html

As for guidance for retrieving the ASTER data we refer to the online tutorial:

- http://www.echo.nasa.gov/reference/astergdem_tutorial.htm

ASTER data conversion

Once the user has downloaded the digital elevation data for his/her region of interest, the data needs to be converted to the GEOGRID binary format. The ASTER data come in tiles that are 1°×1° in size. Consequently, as long as the high resolution (i.e. less than 1km horizontal resolution) domains are located within the ASTER tile in question, the data conversion is straight forward. The data can be converted directly with the convert_geotiff.x utility from the GeoTIFF[1] package.

However, there may be instances where the region of interest lies on the borders of up to four ASTER tiles, assuming that the region of interest is not larger than 1°×1°. To convert the ASTER data to the GEOGRID binary format the user needs to resort to using specific software to handle the conversion. GRASS[2] (Geographic Resources Analysis Support System) is an example of an open source tool that can be used for this purpose. The technique of conversion is to define an area of interest in GRASS, with the spatial resolution equal to that of the ASTER tiles. The tiles are then imported into GRASS and spliced together into a single map. The map is then exported from GRASS in a number of segments (the number depending on map size), and these segments are then converted to the GEOGRID binary format.

Issues regarding ASTER and Corine data

Due to the different resolutions between the ASTER and Corine data there are inconsistencies between terrain height (data from ASTER) and landmask and land-use (Corine data) definitions. This will result in grid points near the coast to be defined as "sea" but have height values greater than zero. The python program ModifyGeoem.py (which is run after the geogrid.exe step) coordinates the topography data and the landmask in the three innermost geo_em.d0X.nc files. One problem was that during the real.exe step, the landmask was converted back to the original values, based on values of the lu_index variable. It is not possible to use the existing geogrid.exe software to interpolate the lu_index variable so it fits the modified landmask data. Furthermore, the metgrid.exe program does not handle this kind of interpolation as it only deals with data on the intermediate format and not the static data handled by geogrid.exe.

A solution to this resolution inconsistency was to write the ModifyMetem.py program that changes the values of lu_index in accordance with the new landmask variable. This modification is more complex than that of the landmask variable, which only has the value of 0 (sea) or 1 (land), as there are many land-use categories. As a first step the code uses "fill" interpolation and "most popular" interpolation for points that have no adjacent non-sea points. A preferable solution would of course be for the WRF community to

---

[1] http://remotesensing.org/geotiff/faq.html and http://www.openwfm.org/wiki/How_to_convert_data_for_Geogrid

[2] http://grass.fbk.eu/

coordinate the two data sets (ASTER and Corine) so individual users would not have to worry about this.

Acknowledgement

The authors acknowledge the assistance of Marius O. Jonassen, both as a willing test user of the system and also for providing a functional version of the Corine data for Norway and Denmark. Discussions with Dr. Idar Barstad over the course of this work further improved the overall structure and quality of the software solution.

Appendix

Model level data from the ECMWF can be interpolated to pressure levels by using the CDO[3] software:

```
#!/bin/bash
#
export EXTRAPOLATE=1
PRESSLEVELS="1000,2000,3000,4000,5000,6000,7000,8000,9000,10000,
15000,20000,25000,30000,35000,40000,45000,50000,55000,60000,
65000,70000,75000,80000,82500,85000,87500,90000,92500,95000,
97500,100000"
#
# find all relevant files and loop through them
#
files=`ls *-ml`
#
# and now loop through
#
for i in $files
do
# use the CDO routines
 echo $i $i"2pl"
 cdo ml2pl,${PRESSLEVELS} $i $i"2pl"
done
```

In addition to the regular surface data from ECMWF, the user needs to have the geopotential height available at least at one pressure level, in addition to the pressure level data created with CDO. The geopotential height can than be interpolated using a patched version of the `ungrib.exe` program. To be more exact, the following patch needs to be added to the `rrpr.F` program under the `ungrib/src` directory:

```
!
! Changes by HA, 4/9 '07
!
! If upper-air HGT is missing integrate from TT and P but currently we
! need a priori the HGT at one pressure level, e.g. at 850 hPa.
!
      do k = nlvl,1,-1
        if ( (.not. is_there(nint(plvl(k)),'HGT') ) .AND. &
             plvl(k) .lt. 200000. .AND. &
             ( is_there(nint(plvl(k+1)), 'HGT')) ) THEN
          call get_dims(nint(plvl(k+1)), 'HGT')
          call compute_hgt_tp(plvl, maxlvl, k, map%nx, map%ny)
        endif
      enddo
```

[3] https://code.zmaw.de/projects/cdo

```
      do k = 1, nlvl
        if ( (.not. is_there(nint(plvl(k)),'HGT') ) .AND. &
             plvl(k) .lt. 200000. .AND. &
             ( is_there(nint(plvl(k-1)), 'HGT')) ) THEN
          call get_dims(nint(plvl(k-1)), 'HGT')
          call compute_hgt_tp(plvl, maxlvl, k, map%nx, map%ny)
        endif
      enddo
!
! Changes by HA done
!
```

And at the end of the program, add this:

```
!
! Changes by HA, 4/9 '07
!
! If upper-air HGT is missing integrate from T and P but currently we
! need a priori the HGT at one pressure level, e.g. at 850 hPa.
!
subroutine compute_hgt_tp(plvl, maxlvl, k, ix, jx)
 use storage_module
 implicit none
 integer :: ix, jx, k, maxlvl
 real, dimension(maxlvl) :: plvl
 real, dimension(ix,jx) :: tt, tto, hgt

 ! Constants
 real, parameter :: Rd=287.05
 real, parameter :: go=9.80665

 call  get_storage(nint(plvl(k)), 'TT', tt, ix, jx)

 if ( (.not. is_there(nint(plvl(k)),'HGT')) .and. &
        ( is_there(nint(plvl(k-1)), 'HGT')) ) then
   call  get_storage(nint(plvl(k-1)), 'TT', tto, ix, jx)
   call  get_storage(nint(plvl(k-1)), 'HGT', hgt, ix, jx)
   hgt = hgt + ( (Rd/go) * 0.5 * (tt+tto) * log(plvl(k-1)/plvl(k)) )
 else  if ( (.not. is_there(nint(plvl(k)),'HGT')) .and. &
        ( is_there(nint(plvl(k+1)), 'HGT')) ) then
   call  get_storage(nint(plvl(k+1)), 'TT', tto, ix, jx)
   call  get_storage(nint(plvl(k+1)), 'HGT', hgt, ix, jx)
   hgt = hgt - ( (Rd/go) * 0.5 * (tt+tto) * log(plvl(k)/plvl(k+1)) )
 endif

 write(*,'("Integrating to fill in HGT at level ", I8, I8, &
   " HGT  ", F30.10)') nint(plvl(k)), k, hgt(ix/2,jx/2)

 call  put_storage(nint(plvl(k)), 'HGT', hgt, ix, jx)

end subroutine compute_hgt_tp
!
! Changes by HA done
!
```

Following is a guideline script to convert multiple ASTER tiles to GEOGRID binary format. Unfortunately, the behaviour of GRASS is both version and platform dependent:

```
#
# This is the method in COMMAND LINE to import and convert the ASTER data....
#
# Let's assume that we have only 6 tiles: ASTGTM_N51E005_dem.tif, ASTGTM_N51E006_dem.tif,
ASTGTM_N50E004_dem.tif, ASTGTM_N50E005_dem.tif, ASTGTM_N49E003_dem.tif and
ASTGTM_N49E004_dem.tif
#
# Start up GRASS and use the "Location wizard" (which isn't a wizard) to create a new location
based on one _dem tile. For the rest, use the GRASS command line window.
#
```

```
# first set the geometry. This is gotten from the tileset names - the n= and s= parameters are the
north and south extents, and the e= and w= parameters are the east and west extents. The n= and
e= extents should be +1 for both parameters.
#
g.region n=52:0:0N s=49:0:0N e=6:0:0E w=3:0:0E res=8.333334e-04

#
# Now we input the tileset and set a few variables...
cd /where/ever/the/tiles/are
tilelist=""
for tile in *_dem.tif
do
  outname=$(echo $tile | sed 's/.tif$/@PERMANENT/g')
  r.in.gdal input=$tile output=$outname
  tilelist=$tilelist+","+$outname
done
tilelist=$(echo $tilelist | sed 's/^,//')
mapname="NORMAP"

#
# Now we need to patch the tiles
r.patch input=$tilelist output=$mapname

#
# and lastly, we want to output the converted tiles. We do however NOT want that to go into the
input file directory, so
# we'll cd to someplace else
cd /someplace/else
g.region n=52:0:0N s=50:0:0N e=6:0:0E w=3:0:0E
r.out.gdal input=$mapname@PERMANENT type=Int16 ouptut=Layer_1
g.region n=50:0:0N s=49:0:0N e=6:0:0E w=3:0:0E
r.out.gdal input=$mapname@PERMANENT type=Int16 ouptut=Layer_2

# this concludes the export.
#
# Now, the conversion has to happen
for x in 1 2
do
  mkdir $x
  pushd $x
  convert_geotiff.x -t 27000 ../Layer_$x
  popd
done
```